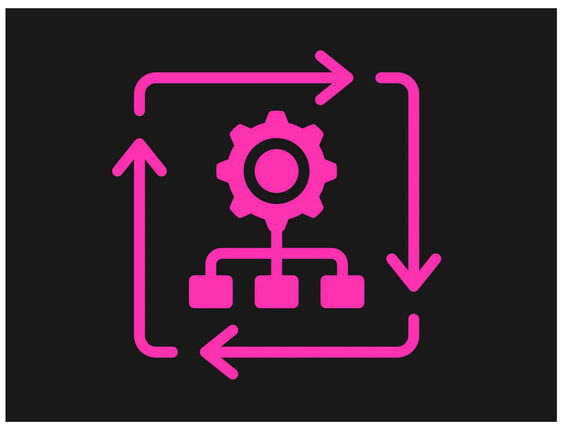
Enabling the Optimal Serverless Platform Team — CDK and Team Topologies

Ben Ellerby · Follow

Published in Serverless Transformation · 7 min read · Aug 10, 2022



We want our Cloud teams to be secure, fast, stable, reactive, autonomous and of course… happy & fulfilled! However, this isn't always the case…

Serverless, and related technologies, have enabled teams to move faster, reduce total cost of ownership and overall empowered developers to have greater ownership of the systems they build. However, Serverless is not a silver bullet — there is an organisational side that's key to unlock the full benefits of Cloud.

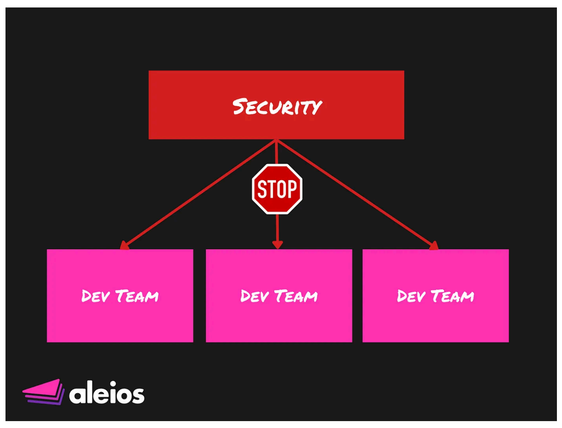
When it comes to team structure as we adopt more Cloud-Native technologies, like Serverless, there are 2 key areas that need to be rethought:

- The top-down approach to security bottlenecking developers
- Overwhelming *cognitive load*

As discussed above, the solution is not purely technical — but CDK can enable the socio-technical change needed.

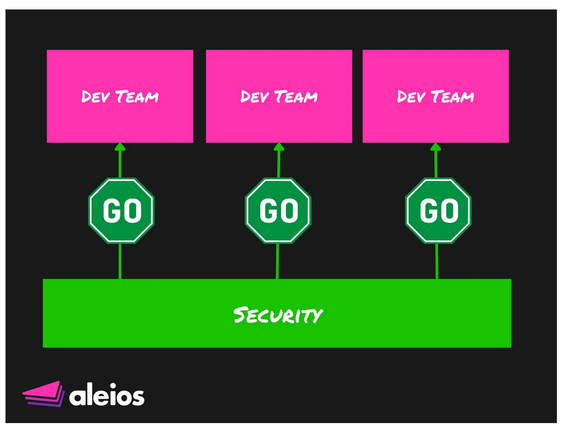
Rethinking Security

In many large organisations that move to adopt Serverless, security is typically seen as a top-down bottleneck that slows development and innovation. At the same time security incidents have been rising for years, and a strong cyber security posture requires organisations to work together, not just *throwing things over the wall* to *InfoSec*.



Security as "Top Down" Blocker

As companies have started to adopt Cloud and DevOps developers have been brought closer to security, yet a wall still exists in this collaboration. With Serverless the application code and the cloud are inextricably linked, meaning developers have moved further into the realm of the security. This typically creates some turf wars in the initial stages of a transformation to adopt Serverless, but also provides the opportunity for us to rethink the traditional security approach.

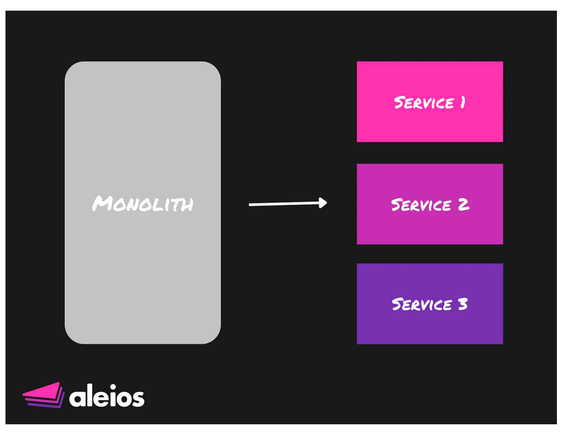


Ideally Security is seen as an enabling function, with tools, approaches and teams enabling a secure by default paradigm. This requires us to turn our view of security on its head and have development teams be enabled by security rather than held hostage from deploying or innovating.

This requires a change in engineering team structure, as well as a technical solution to enable a bottom-up secure by design paradigm. The combination of "Team Topologies" and CDK can be leveraged to achieve this change.

Cognitive Load

Cognitive load is a form of friction on teams — the greater the cognitive load the slower their velocity (and often the lower their satisfaction). One key way to reduce cognitive load is to design our architecture and systems on the underlying domain and building services based on independent subsections of the domain to enable independence of teams and services. This allows teams to restrict their scope to a subset of the overall domain complexity.



> ***Cognitive Load:*** *The amount of working memory being used.*
> *Team Topologies, Matthew Skelton & Manuel Pais*

As cloud and application code move closer together we're leveraging smaller cross-functional teams to a greater and greater degree. This has many advantages, with smaller teams building at a higher velocity and with complete autonomy. At the same time, if we don't design our architectures and organisational structures correctly it can create such a large cognitive load on the teams involved that they struggle to deliver and it becomes extremely difficult to onboard new developers.

Team Topologies, the book and framework from Matthew Skelton & Manuel Pais, proposes a very handy way to structure teams and leverage the *Inverse Conway Manoeuvre* to design our technical architecture and organisational structure together.

Inside team topologies there are 4 types of team:

1. **Stream-Aligned Team:** Ideally split into segments of the Business Domain (e.g. Bounded Contexts) and delivering a flow of work.
2. **Enabling Team:** Assist Stream-Aligned teams, remove obstacles and detect lacking capabilities.
3. **Complicated Subsystem team:** Domain specific complex expertise (e.g. algorithmic, machine learning)
4. **Platform Team:** Accelerate the Stream-Aligned teams via a product based approach while maintaining their autonomy.

In combination with Domain-Driven Design techniques Stream-Aligned Teams can work on independent areas of the domain with services architected accordingly (see EventBridge Storming).

With a Serverless approach much of the complexity of building, managing and running applications at scale is abstracted away. However, developers are now exposed to distributed tracing, service limit throttles, observability, security, accounts, cross-account management…

Platform Teams can greatly reduce the cognitive load placed on Stream-Aligned teams by providing foundational services that no longer need to be created by the Stream-Aligned teams. These Platform teams should see themselves as service providers, with the Stream-Aligned teams as their "customers".

In the case of Serverless development a Platform team could enable autonomous Stream-Aligned teams to work with higher level abstractions over the lower level Cloud provider services (we'll focus on AWS in this article, but a similar approach can be taken with other providers). This can remove complexity in getting APIs up and running, as well as provide security defaults that match the needs of the organisation.

For instance, many companies have a, very sensible, policy against public S3 buckets and also rules on the encryption and retention policies of these storage solutions. Rather than expose every team to the cognitive load of these defaults a Platform team could instead provide a higher order abstracted component to enable Stream-Aligned teams to work safely and quickly.

CDK — The Enabling Technology

It's clear that team structure is one part of the puzzle to solving the issues of security bottlenecking and overwhelming cognitive-load for companies adopting a Serverless-First approach. This approach though needs an appropriate enabling technology to enable a Platform team to provide a simple self-service collection of pre-configured and abstracted Cloud constructs.

Key to enable the platform team is the need for:

- Abstraction
- Encapsulation
- Composition

These components would enable the Platform team to hide complexity via abstraction, provide protected security defaults via encapsulation and allow Stream-Aligned Teams to self-compose components into their sub-domain specific use cases.

What is CDK?

> *The AWS Cloud Development Kit (AWS CDK) is an open-source software development framework to define your cloud application resources using familiar programming languages.*
> Amazon Web Services

AWS CDK allows developers to use their programming language of choice, e.g. TypeScript, to define their architecture. It allows developers to use the powerful expressive characteristics of modern programming languages, including concepts like **Abstraction, Encapsulation & Composition**.

Under the hood CDK synthesises familiar CloudFormation, providing a repeatable deployment and compatibility with existing deployment tooling and practices.

The Construct Hierarchy

CDK Constructs represent the basic building blocks of AWS architectures. They contain everything needed to generate the CloudFormation needed to deploy a part of the architecture.

There are 3 levels of construct, each building on each other:

- L1. **Level 1:** AWS CloudFormation-Only
- L2. **Level 2:** Curated
- L3. **Level 3:** Patterns

L1 constructs are generated directly from CloudFormation, and therefore represent a feature complete interface to generate resources at the lowest level of abstraction. L2 constructs build on L1, adding some default values, boilerplate and glue. And finally L3 constructs are patterns for common use cases, e.g. a REST API, often composing multiple types of resources into an end-to-end pattern.

L1 constructs are a feature complete and auto-generated periodically from the underlying CloudFormation definitions.

```
const bucket = new s3.CfnBucket(this, "MyBucket", {
  bucketName: "MyBucket"
});
```

L2 constructs build on L1, adding some default values, boilerplate and glue.

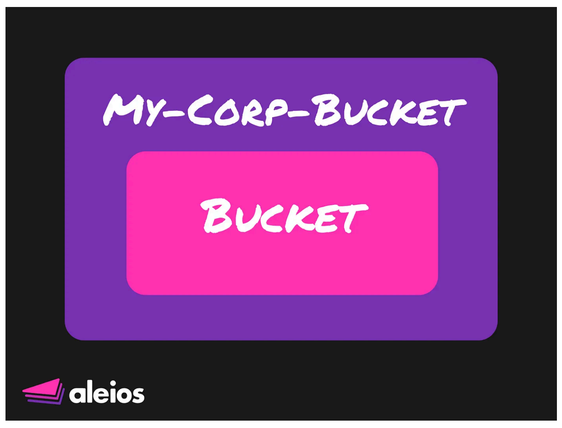
```
import * as s3 from 'aws-cdk-lib/aws-s3';

// "this" is HelloCdkStack
new s3.Bucket(this, 'MyFirstBucket', {
  versioned: true
});
```

And finally L3 constructs are patterns for common use cases, e.g. a REST API, often composing multiple types of resources into an end-to-end pattern.

Baked-In Best Practices — L2+

L2 Constructs are built by the AWS CDK team and typically encapsulate L1 constructs, adding sensible best-practice defaults. A platform team can take this a step further by wrapping these L2 construct with further built-in defaults and best-practices that represent their organisational specific requirements.



For instance, a company in a regulated industry may have rules encryption standards or data retention policies. The Platform team could create a `MyCorpBucket` that adds in the correct encryption, privacy and retention policy by default. This approach abstracts complexity, encapsulates sensible defaults (e.g. bucket privacy) and can be leveraged via composition in use-case specific implementations by the Stream-Aligned Teams.

Not a Compliance Panacea

The L2+ approach can ease the cognitive load on Stream Aligned Teams and bring security considerations to the surface early in the development process. However, they do not ensure compliance.

Control Tower, permissions boundaries, service control policies and other "top-down" security governance and safeguard tools are still needed. We should position security as a bottom-up enabling function via the combination of the correct team topology and expressive IaC tooling like CDK. This should be coupled with appropriate top-down controls to detect issues, prevent mistakes and find opportunities for the Platform team to address and improve.

In Conclusion

Taking full advantage of Serverless requires an organisational, as well as technical approach. We need to address the issues of top-down security bottlenecking and overwhelming cognitive load to unlock the advantages a Serverless-First approach can bring.

CDK provides the abstraction, encapsulation and composition needed for a Platform team to turn the traditional security paradigm on its head and move to a bottom-up enabling function. This is not a fix-all though, and some top-down security governance is needed to ensure safety.

Serverless-Transformation is an aleios initiative. aleios helps startups disrupt and large organisations to remain competitive using the best of Cloud-Native, **Serverless**.

Serverless · Domain Driven Design · Cdk · AWS · Cloud Migration

Written by Ben Ellerby

662 Followers · Editor for Serverless Transformation

Founder of aleios (https://www.aleios.com/) and AWS Serverless Hero

Follow

More from Ben Ellerby and Serverless Transformation

Ben Ellerby in Serverless Transformation

EventBridge Storming — How to build state-of-the-art Event-Drive...

Serverless Architectures should be Event-Driven and use Amazon EventBridge. Use...

10 min read · Apr 7, 2020

616 · 5

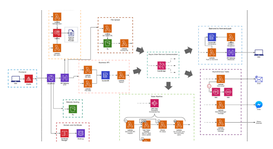
Xavier Lefèvre in Serverless Transformation

Asynchronous client interaction in AWS Serverless: Polling,...

Event-driven Serverless often requires async update to the client. There are several ways ...

6 min read · Apr 18, 2020

730 · 7



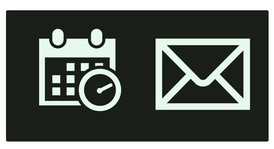
Xavier Lefèvre in Serverless Transformation

What a typical 100% Serverless Architecture looks like in AWS!

If you are new to serverless and looking for a high level web architecture guide, you've...

11 min read · May 19, 2020

2.8K · 19



Ben Ellerby in Serverless Transformation

Serverless Event Scheduling — Using AWS Step Functions

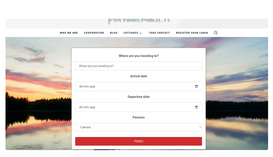
Serverless is not just FaaS. Many serverless services can be combined to build elegant...

6 min read · Oct 28, 2019

188 · 1

See all from Ben Ellerby · See all from Serverless Transformation

Recommended from Medium



Artturi Jalli

I Built an App in 6 Hours that Makes $1,500/Mo

Copy my strategy!

· 3 min read · Jan 23, 2024

Amo Moloko

Migrating from the Serverless Framework to AWS CDK

Serverless as a technology is no longer a fad as we venture into 2024. Aleios who have...

12 min read · 4 days ago