Convince your boss to go Serverless

by Vadym Kazulkin, ip.labs GmbH

Contact

Vadym Kazulkin

ip.labs GmbH Bonn, Germany

Co-Organizer of the Java User Group Bonn and Serverless Bonn Meetup

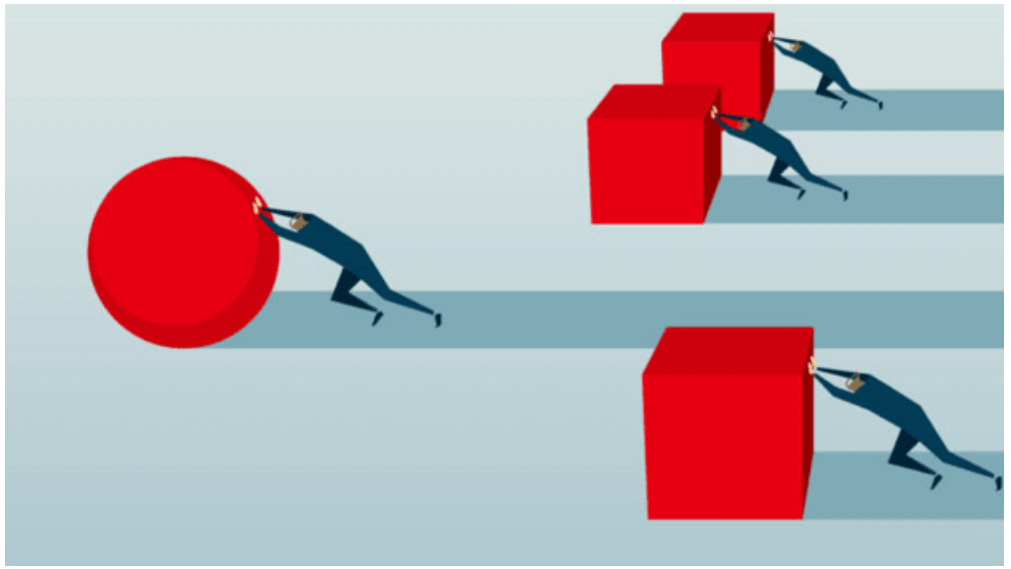
v.kazulkin@gmail.com

https://www.linkedin.com/in/vadymkazulkin/

@VKazulkin
@ServerlessBonn (Meetup)

ip.labs GmbH

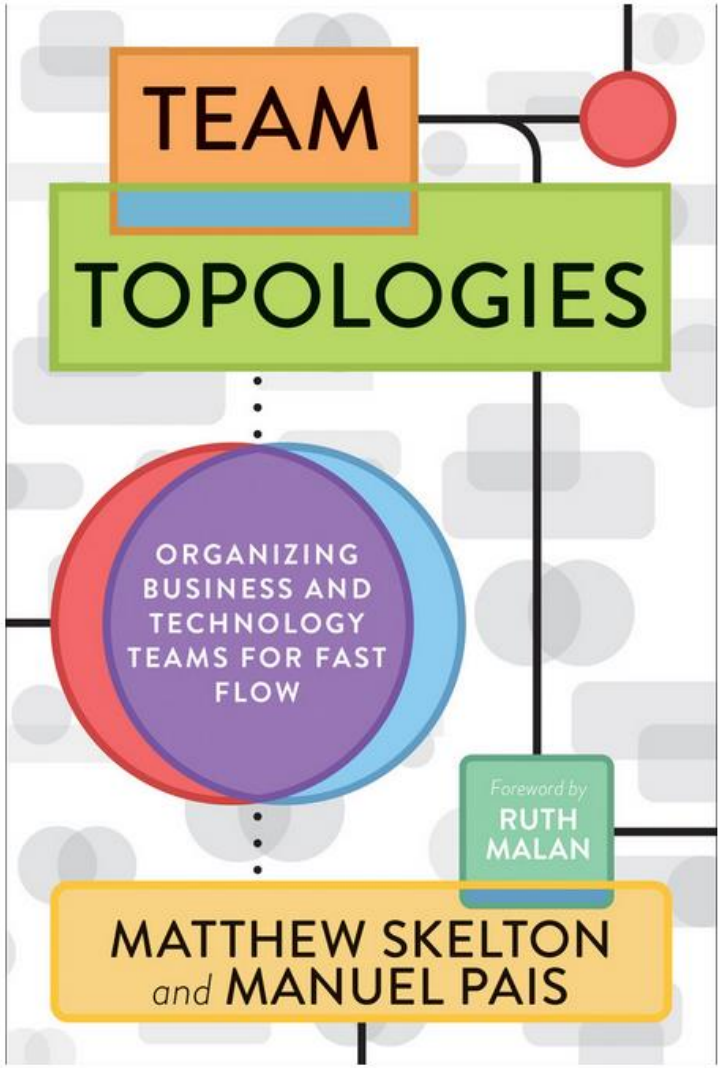
Let’s talk about the **challenges** of the software development in general first



Cognitive Load –

the total amount of mental effort being used in the working memory

- Intrinsic
- Extraneous
- Germane



Cognitive Load

- Intrinsic
  - How to write a Java class or use a framework (Spring)
- Extraneous
- Germane

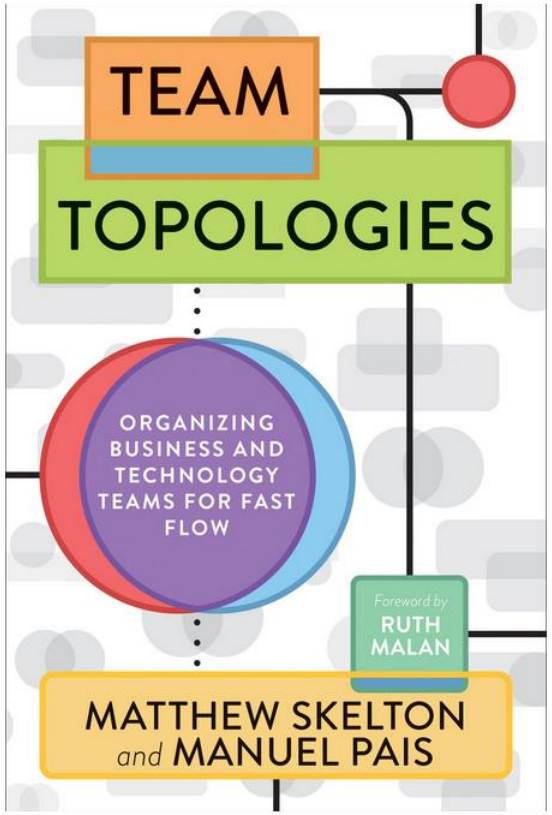
Cognitive Load

- Intrinsic
- Extraneous
  - How to automate tests (unit, integration, end-to-end, web, desktop, mobile)
  - How to build, package, deploy and run my application
  - How to configure monitoring, alerting, auto-scaling, logging and tracing
  - How to operate and maintain infrastructure
  - How to build-in fault-tolerance and resiliency
  - How to make the hardware, networking and application secure
- Germane

Cognitive Load

- Intrinsic
- Extraneous
- Germane
  - Domain Knowledge (payment, e-commerce)
  - Business processes and workflows

Cognitive Load



- Intrinsic -> become fluent in it
- Extraneous -> minimize amount of what we implement/operate/support/own by ourselves
- Germane -> minimize amount of what we have to implement by ourselves

What our boss wants from us?

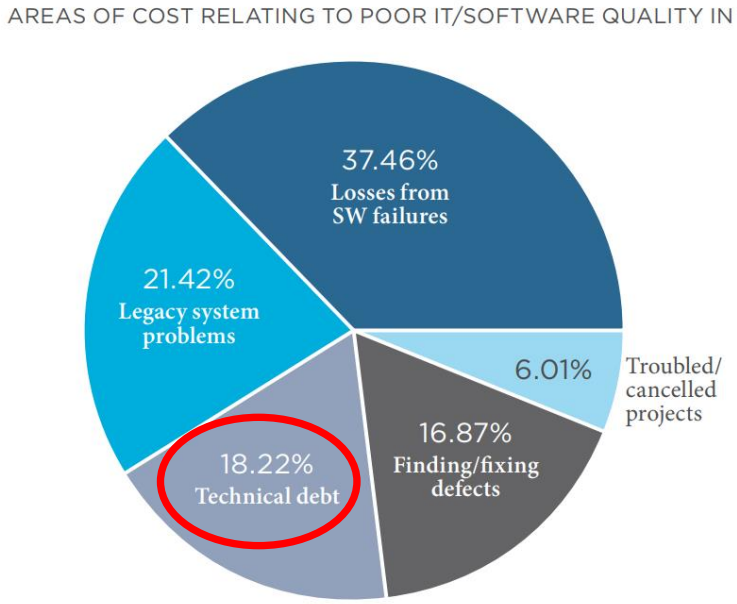
… that we are productive

Productivity

We are productive if we regularly ship products,
which are successfully used by our customers

What is holding us back from being productive?

Technical Debt - reflects the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer



Technical Debt

- Even a perfect solution can become the technical debt over the time
  - Version of programming language comes out of support (Java 8)
  - Security considerations forces us to upgrade one of our dependencies (library or web application server version)
  - One of our dependencies (e.g. to open source project) is discontinued

Technical Debt

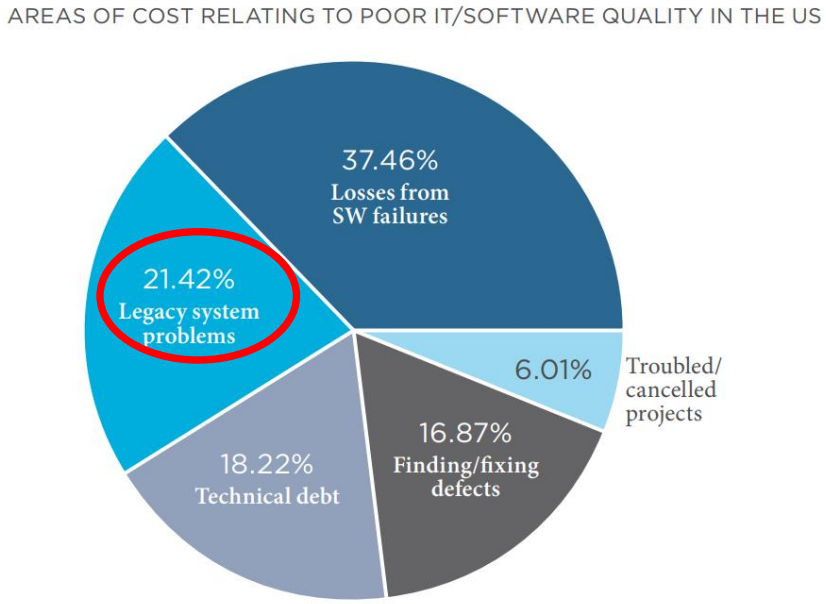
Think of what can happen to your software over
the entire life cycle of our product

Technical Debt

- is related to amount of code written
- is related to amount of dependencies used
  - open source projects, programming languages, databases, (web) application servers

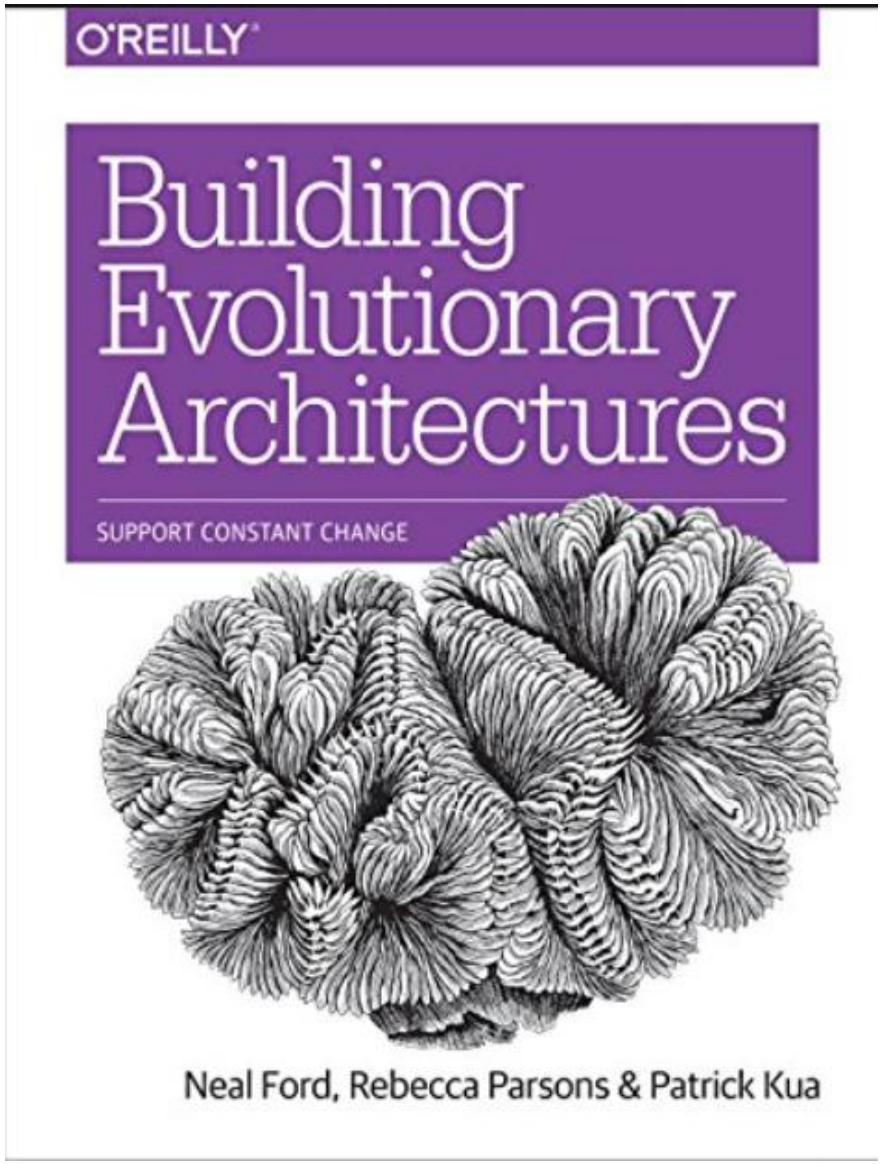
Legacy System

Legacy Systems are systems that can't **evolve**



Evolutionary Architecture –
supports guided, incremental change across multiple dimensions

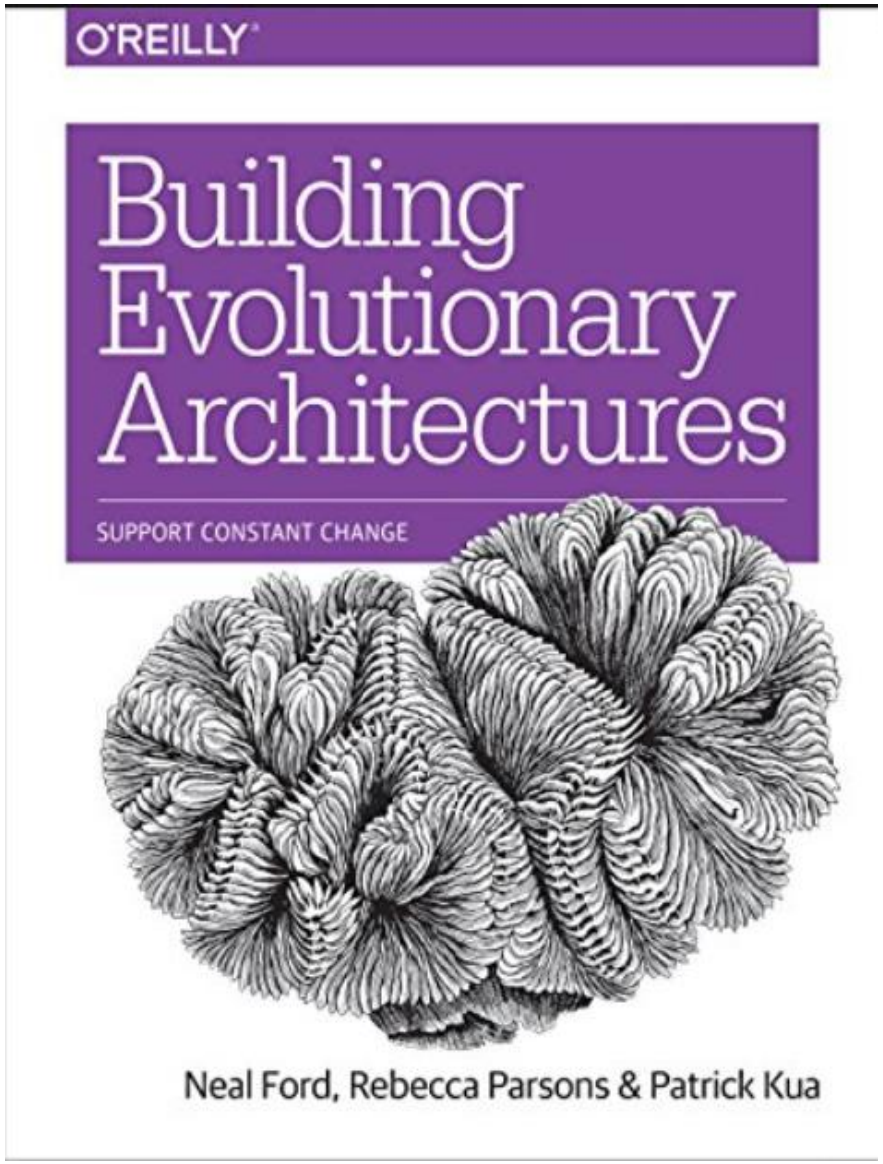
- Incremental change
- Appropriate architectural coupling
- Fitness functions



”Architectural Coupling” https://learning.oreilly.com/library/view/building-evolutionary-architectures/9781491986356/ch04.html

Evolutionary Architecture – Fitness functions

- Source code metrics (such as measuring cyclomatic complexity)
- Unit tests (% of coverage and % of success)
- Performance metrics (such as API latency or throughput)
- Security (encryption at rest, e.g. checking that all S3 buckets have encryption enabled, or automatic key rotation for all external APIs, with tools such as the AWS Secrets Manager)
- ArchUnit, Sonar, CI/CD Tools
  - CodeCommit,…CodeDeploy, Jenkins



The Value Proposition of Serverless

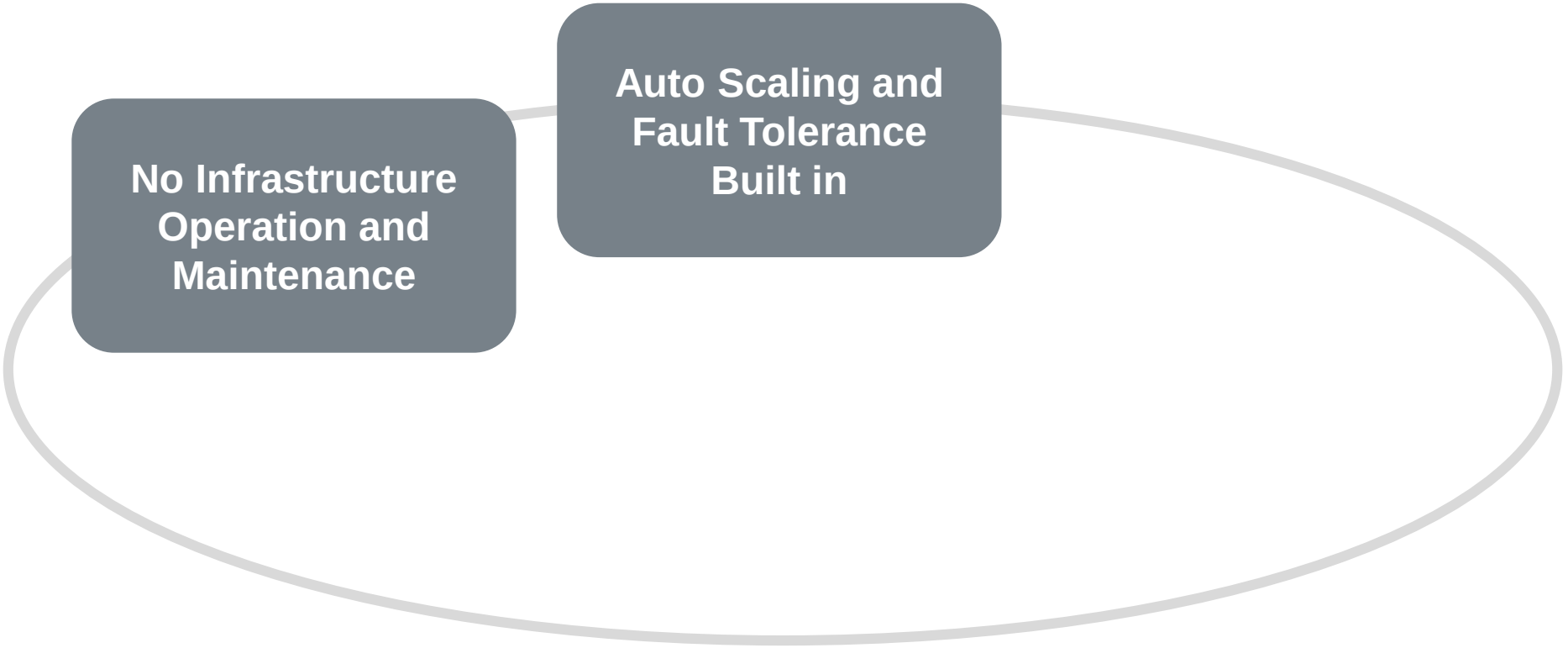
But let’s talk about of Total Cost of Ownership of the Serverless paradigm

TCO Full Picture

No Infrastructure Maintenance

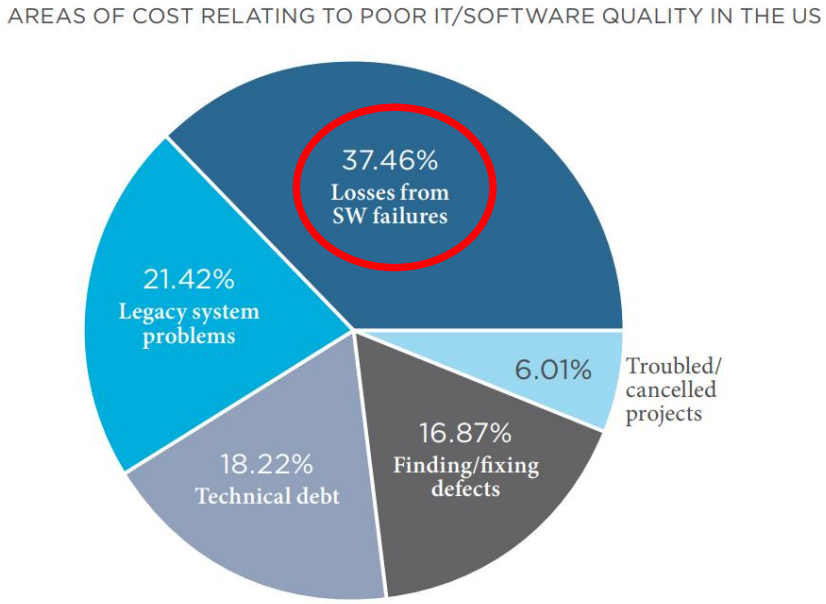
Is infrastructure maintenance and operation your core competency ?

TCO Full Picture

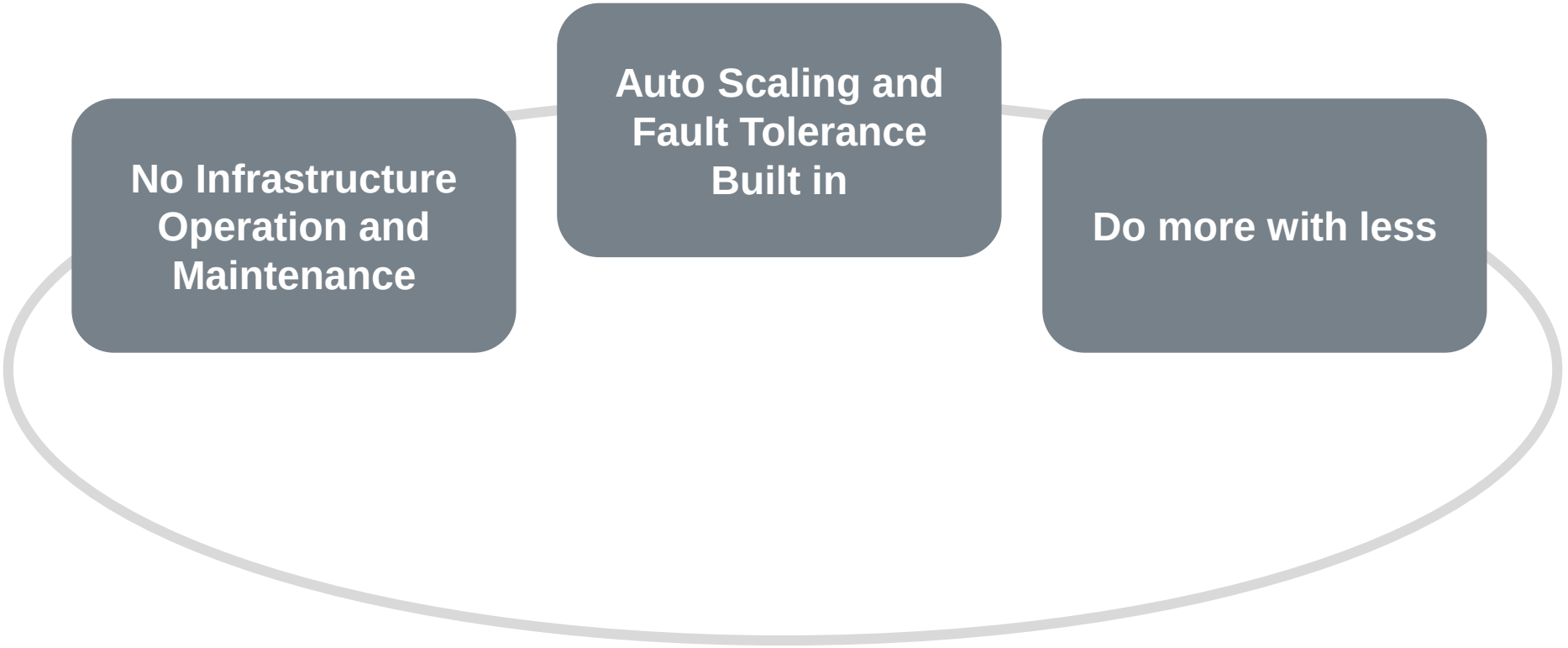


Auto Scaling And Fault Tolerance Built In

- Can you get capacity planning and auto scaling right?
- Do you want to solve the hard problem of fault tolerance by yourself?



TCO Full Picture

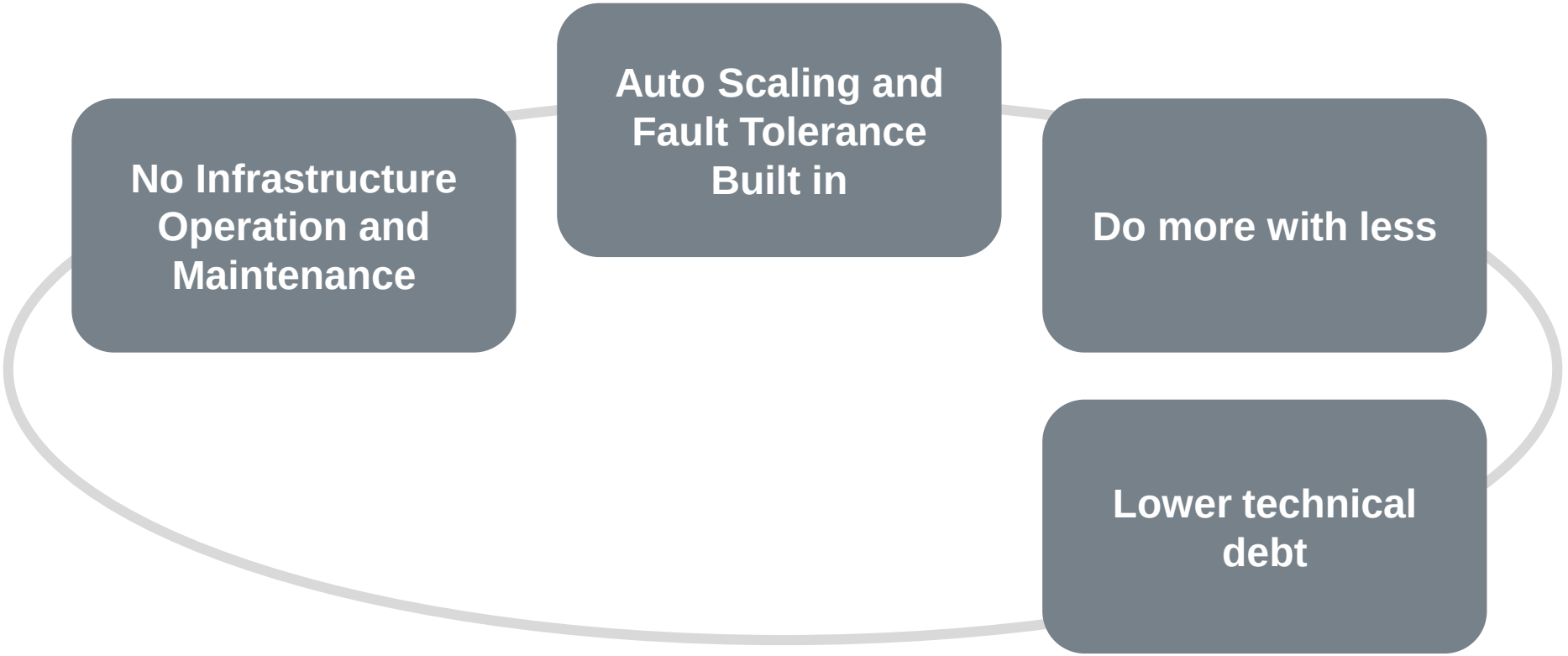


Do more with less

By heavily relying on the managed Serverless services you

- Need fewer engineers to start implementing your new product idea
- Can do more with the same amount of people

TCO Full Picture

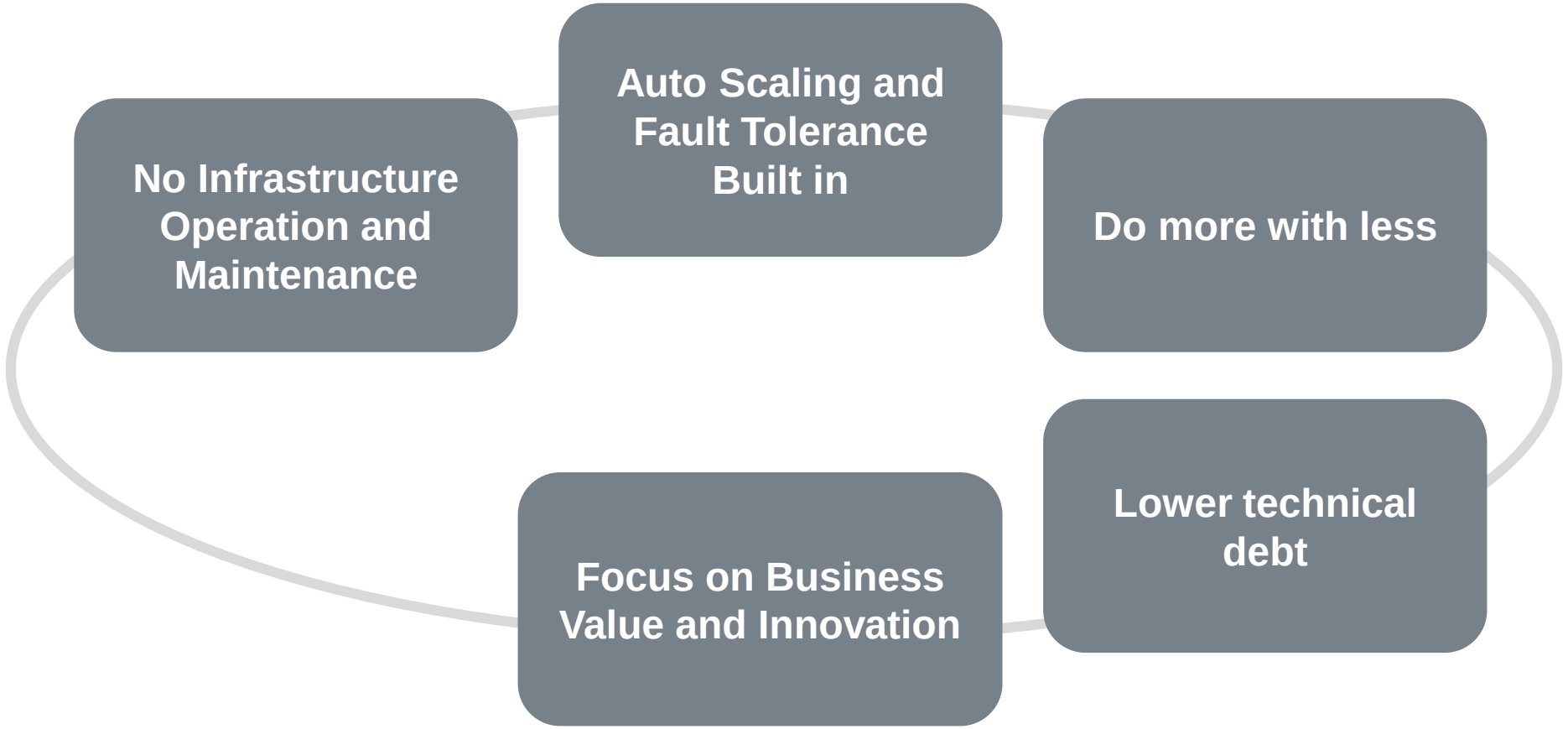


Lower technical debt

- Whatever code you write today is always tomorrow's technical debt © Paul Johnston
- Less code means lower technical debt
- Time and effort required for maintaining the solution over its whole lifecycle is by far much more than for developing it

Jeff Atwood „The Best Code is No Code At All" https://blog.codinghorror.com/the-best-code-is-no-code-at-all/
Paul Johnston "Cloud 2.0: Code is no longer King — Serverless has dethroned it"
https://medium.com/@PaulDJohnston/cloud-2-0-code-is-no-longer-king-serverless-has-dethroned-it-c6dc955db9d5

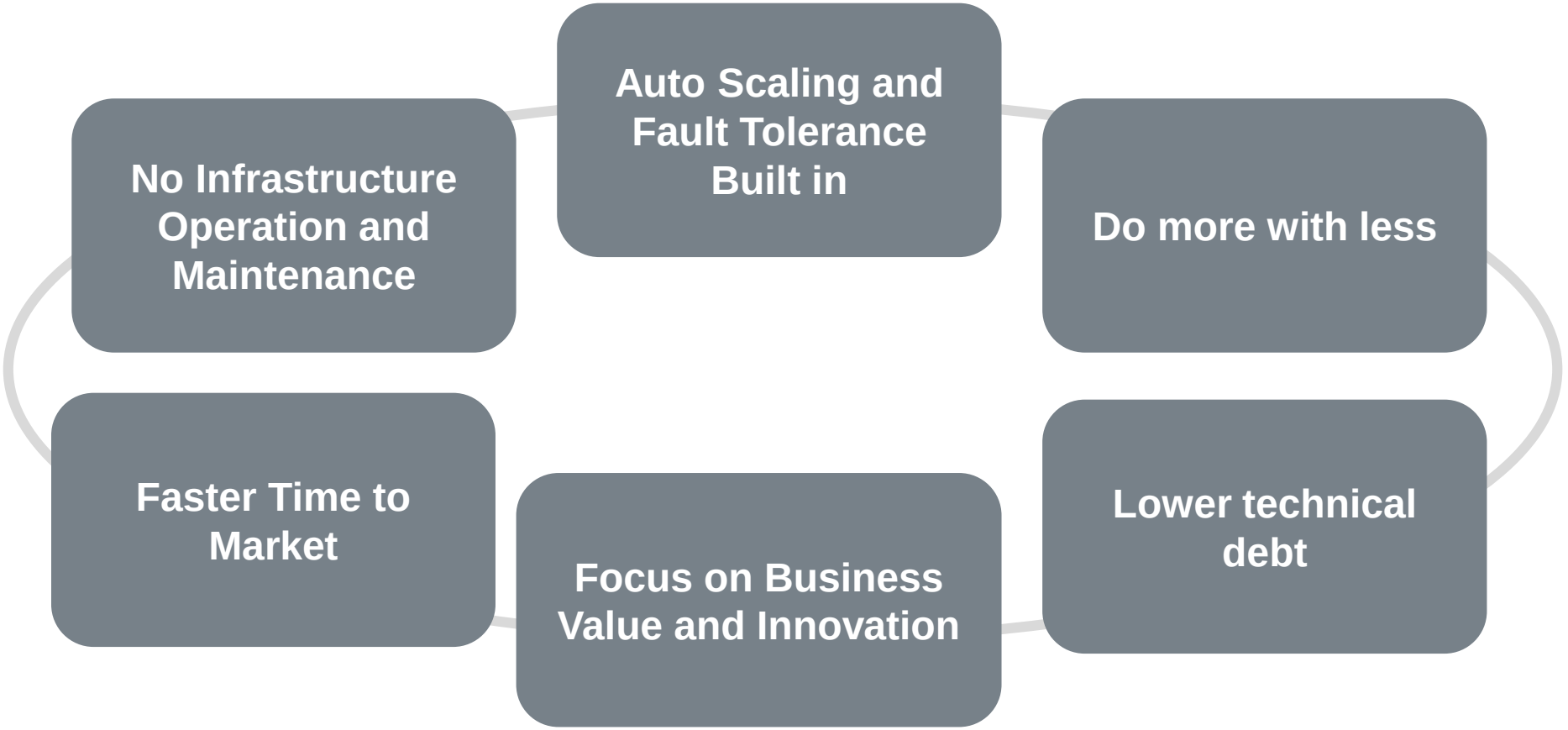
TCO Full Picture



Focus On Business Value and Innovation

Every organization wants exactly this!

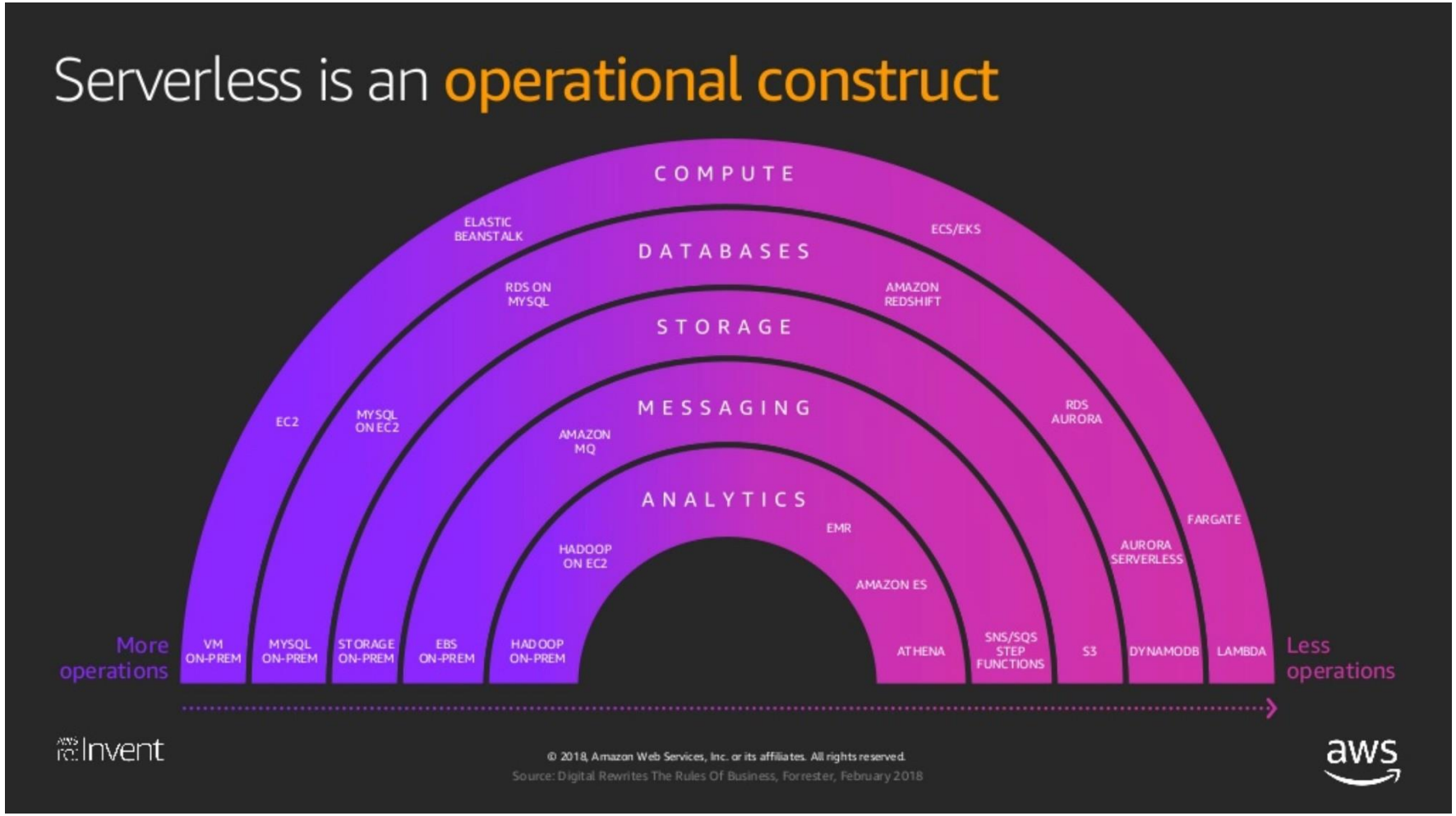
TCO Full Picture



Faster Time To Market

- Time To Market is the key differentiator in today's business!
- Ask yourself: what is core for your business and what you can get as Commodity +(Utility) as a Service?

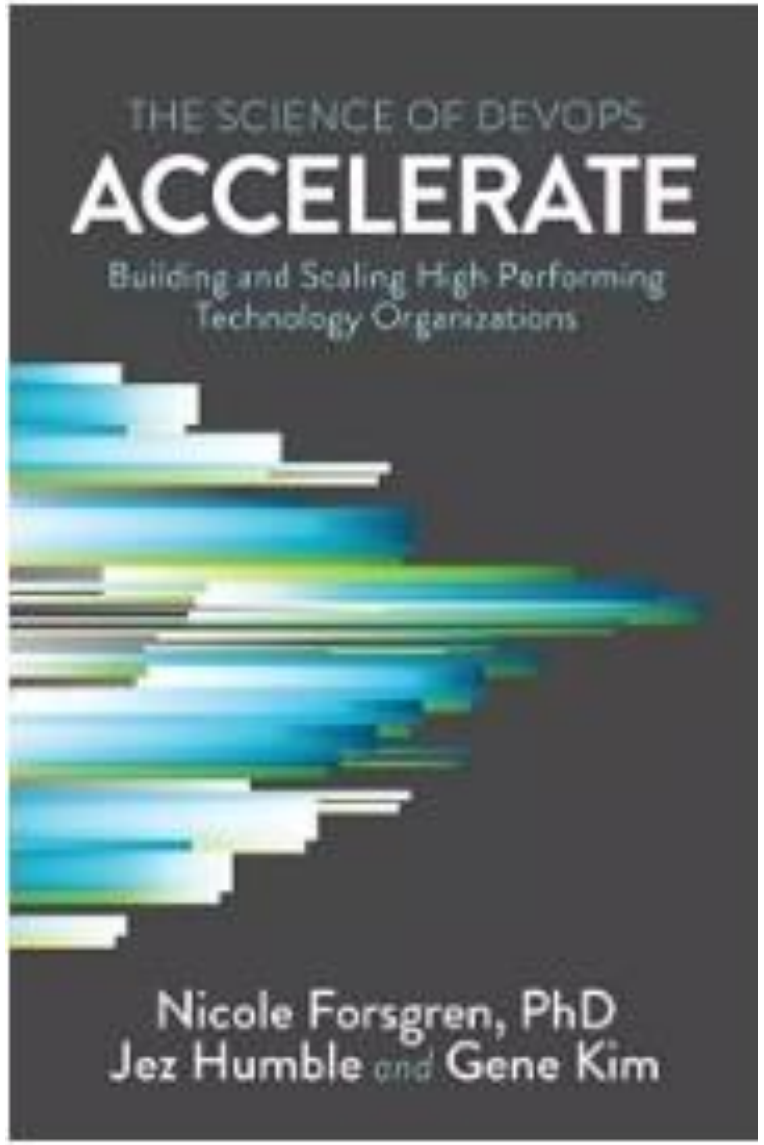
Serverless Mindset at ip.labs



"Accelerate Innovation and Maximize Business Value with Serverless Applications"
https://www.slideshare.net/AmazonWebServices/accelerate-innovation-and-maximize-business-value-with-serverless-applications-srv212r1-aws-reinvent-2018

How to measure success

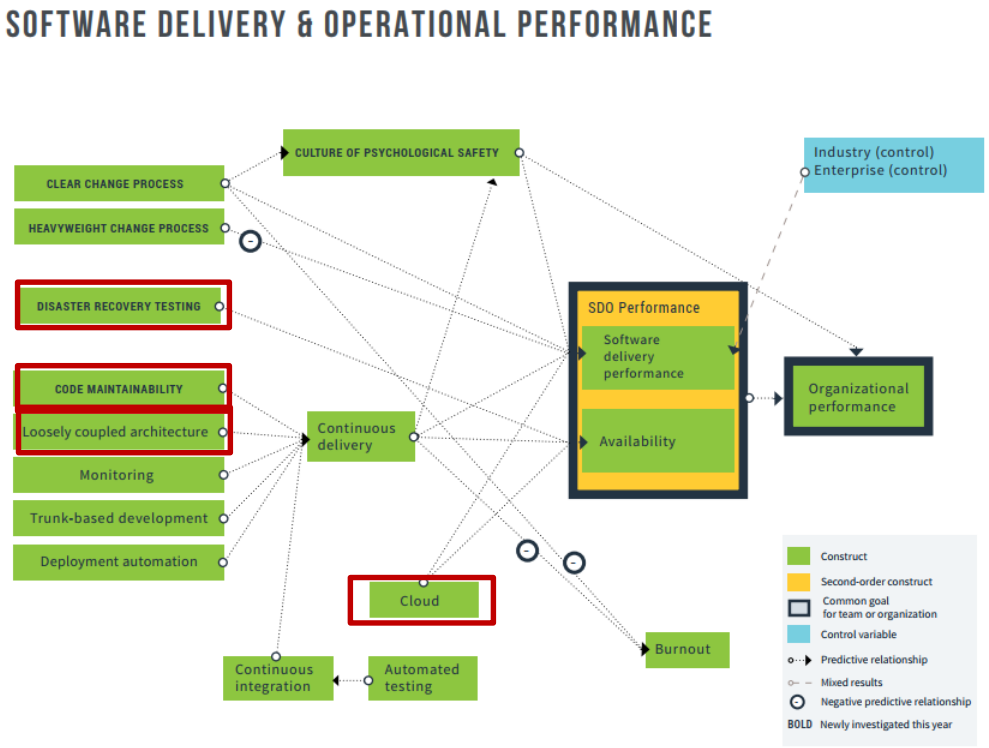
Aspect of Software Delivery Performance*	Elite	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per day and once per week	Between once per week and once per month	Between once per month and once every six months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one day	Between one day and one week	Between one week and one month	Between one month and six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day[e]	Less than one day[a]	Between one week and one month
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0-15%[b,c]	0-15%[a,d]	0-15%[c,d]	46-60%



See DORA State of DevOps 2018-2019 Reports

ip.labs

Software Delivery and Operational Performance



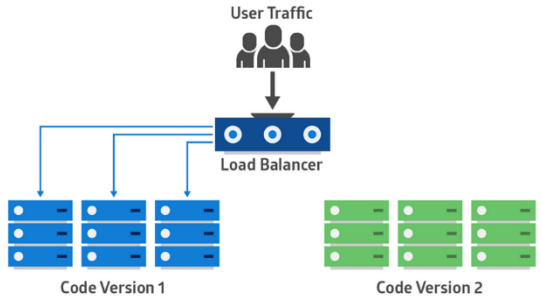
See DORA State of DevOps 2018-2019 Reports

Example: strategies to reduce time to restore service

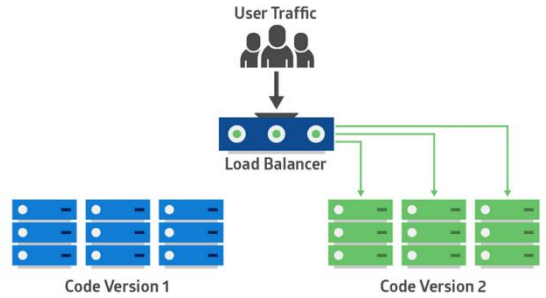
Aspect of Software Delivery Performance*	Elite	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per day and once per week	Between once per week and once per month	Between once per month and once every six months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one day	Between one day and one week	Between one week and one month	Between one month and six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day[a]	Less than one day[a]	Between one week and one month
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0-15%[b,c]	0-15%[b,d]	0-15%[c,d]	46-60%

See DORA State of DevOps 2018-2019 Reports

Blue-Green deployment

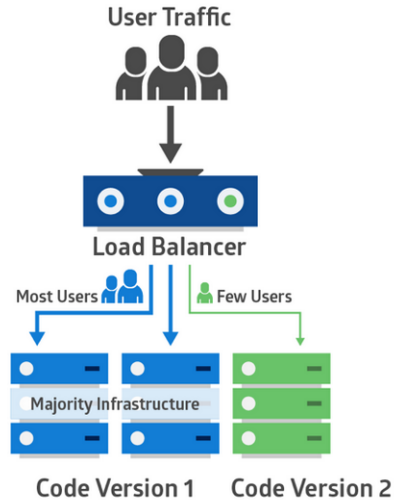


Before a blue-green deployment



After a blue-green deployment

Canary deployment



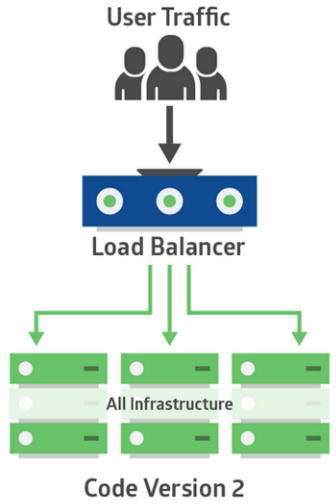
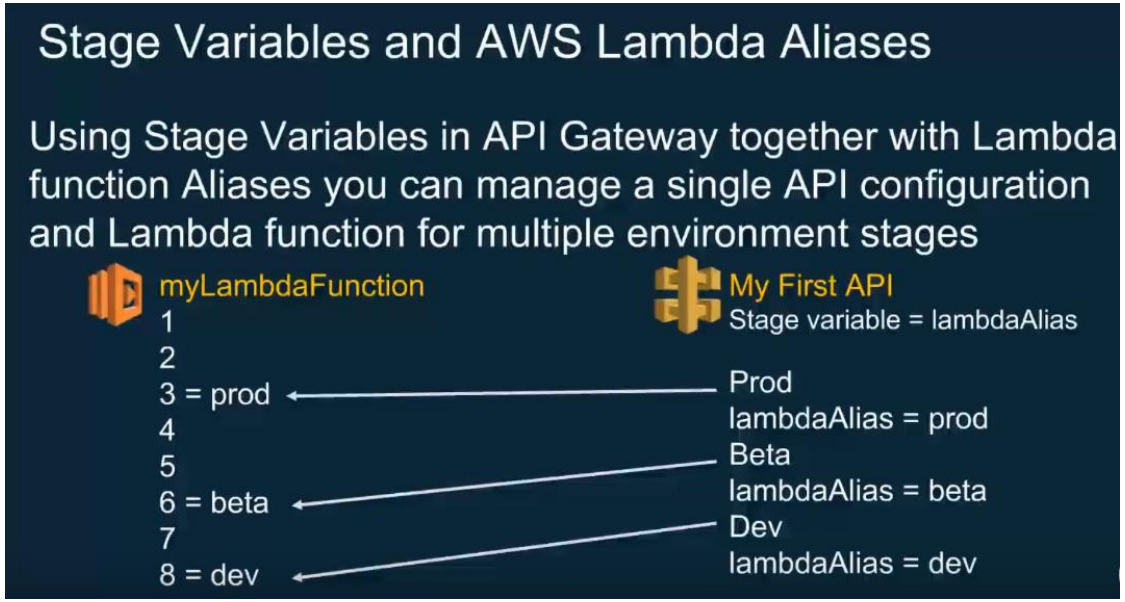
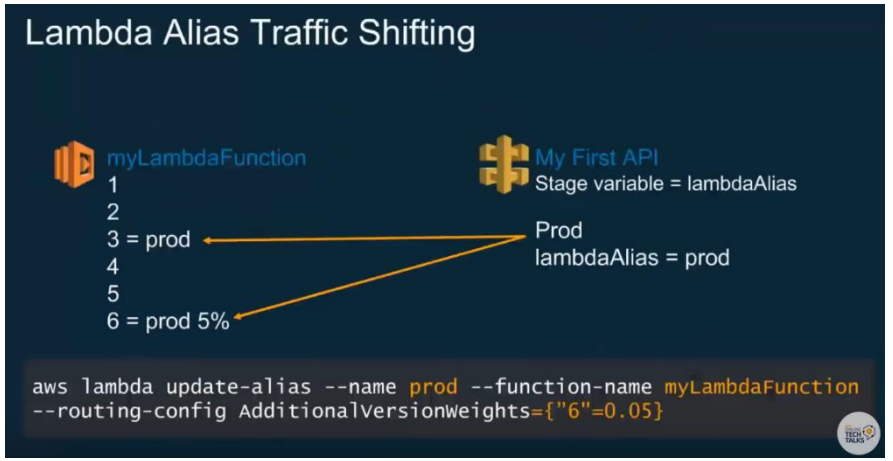


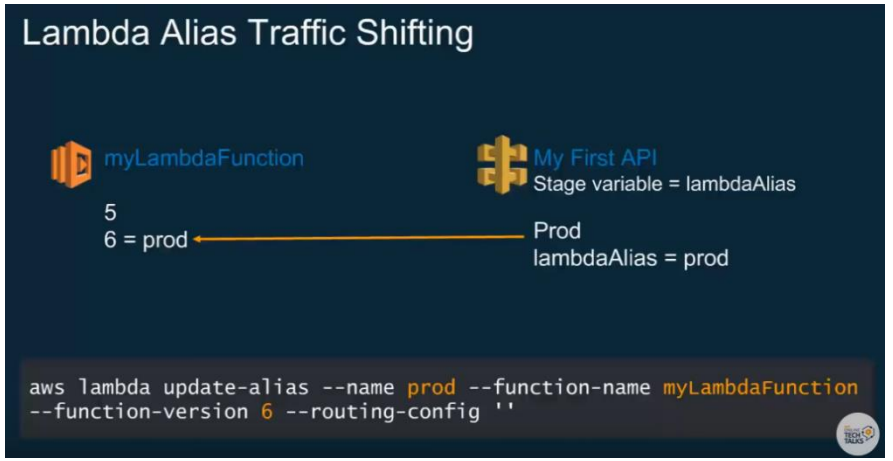
Illustration of a canary deployment

AWS Lambda Deployment Best Practices:

- API Gateway Stage variables and Lambda Aliases
- Lambda Alias Traffic Shifting





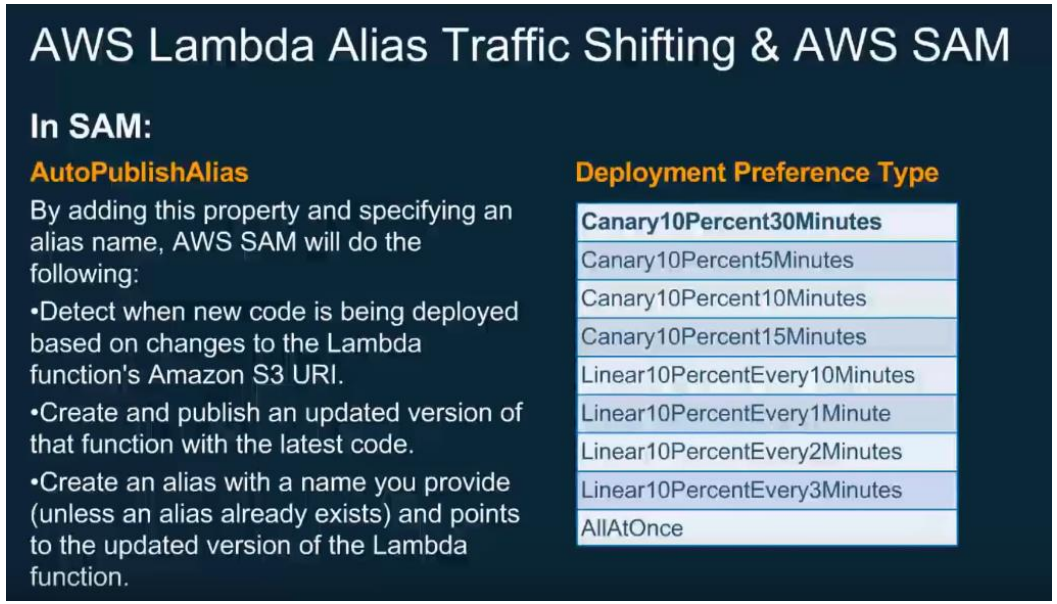


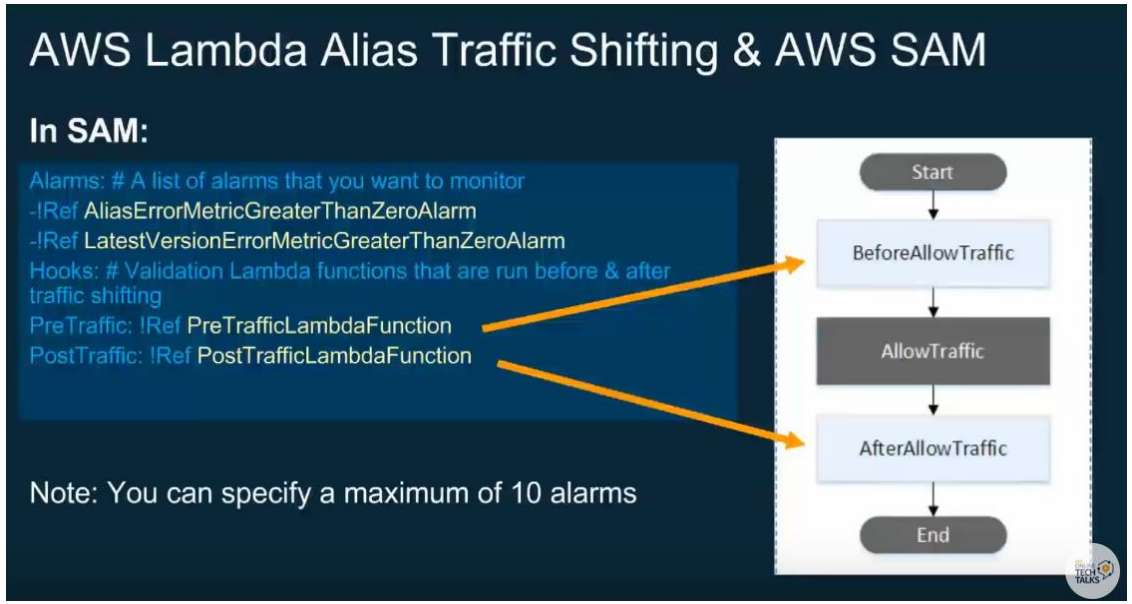
Chris Munns: “Testing and Deployment Best Practices for AWS Lambda-Based Applications”
https://www.youtube.com/watch?v=zJQDAsWm-5k&list=LLYgjRSI2oCzI9eooyFrWR7A&index=1

ip.labs

AWS Lambda Deployment Best Practices

- AWS Lambda Alias *Canary and Linear* Traffic Shifting & AWS SAM Safe Deployments
- CloudWatch Rollback Alarms & Lambda hooks





Chris Munns: “Testing and Deployment Best Practices for AWS Lambda-Based Applications”
https://www.youtube.com/watch?v=zJQDAsWm-5k&list=LLYgjRSI2oCzI9eooyFrWR7A&index=1

ip.labs

Time Spent

Time Spent	Elite	High	Medium	Low
NEW WORK	**50%**	**50%**	**40%**	**30%**
Unplanned work and rework	19.5%	20%[a]	20%[a]	20%[a]
Remediating security issues	5%	5%[b]	5%[b]	10%
Working on defects identified by end users	10%	10%[c]	10%[c]	20%
Customer support work	5%	10%	10%	15%

See DORA State of DevOps 2018-2019 Reports

Using Serverless ecosystem will

with the right engineering practices in place will significantly reduce

- extraneous and germane cognitive load
- the amount of code written

How to write less code with AWS Serverless services 1/3

- Write fewer Lambda functions
  - use direct AWS service integrations in case the Lambda only calls the AWS service itself

Less Lambda functions means less:

- code to write, test, run and maintain
- CI/CD (deploy, rollback strategies) to maintain
- Infrastructure as a Code (IAM policies, permission) to write and test
- cold-start worries
- point of failures and retries
- security concerns
- worries about Lambda limits (e.g. concurrency settings per AWS account)
- spending on Lambda, CloudWatch and 3rd party SaaS on (AWS) (monthly) bill

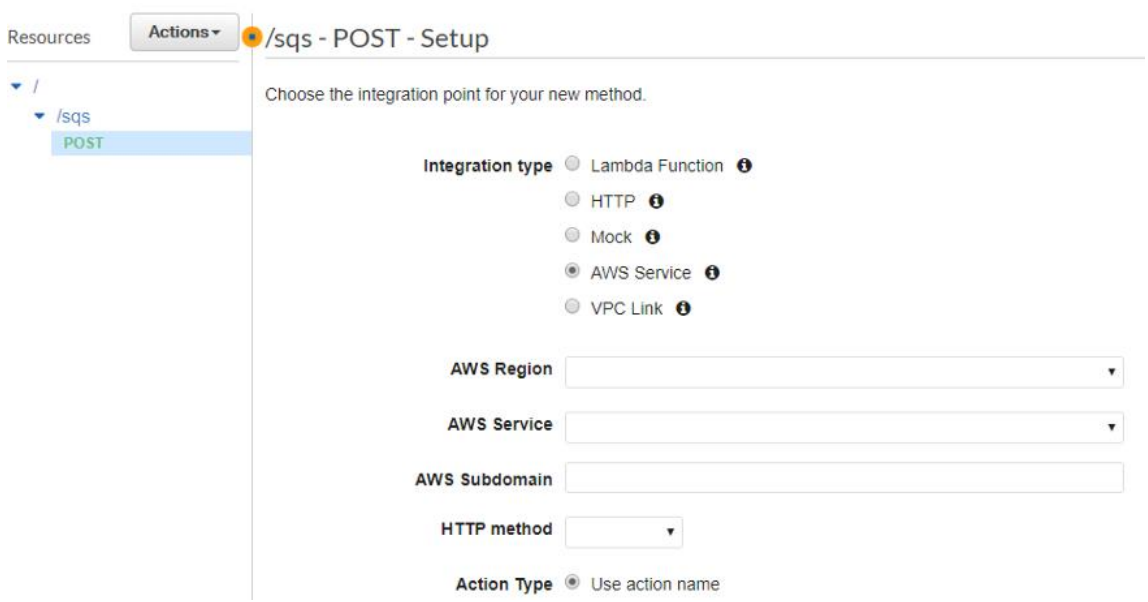
How to write less code with AWS Serverless services 2/3

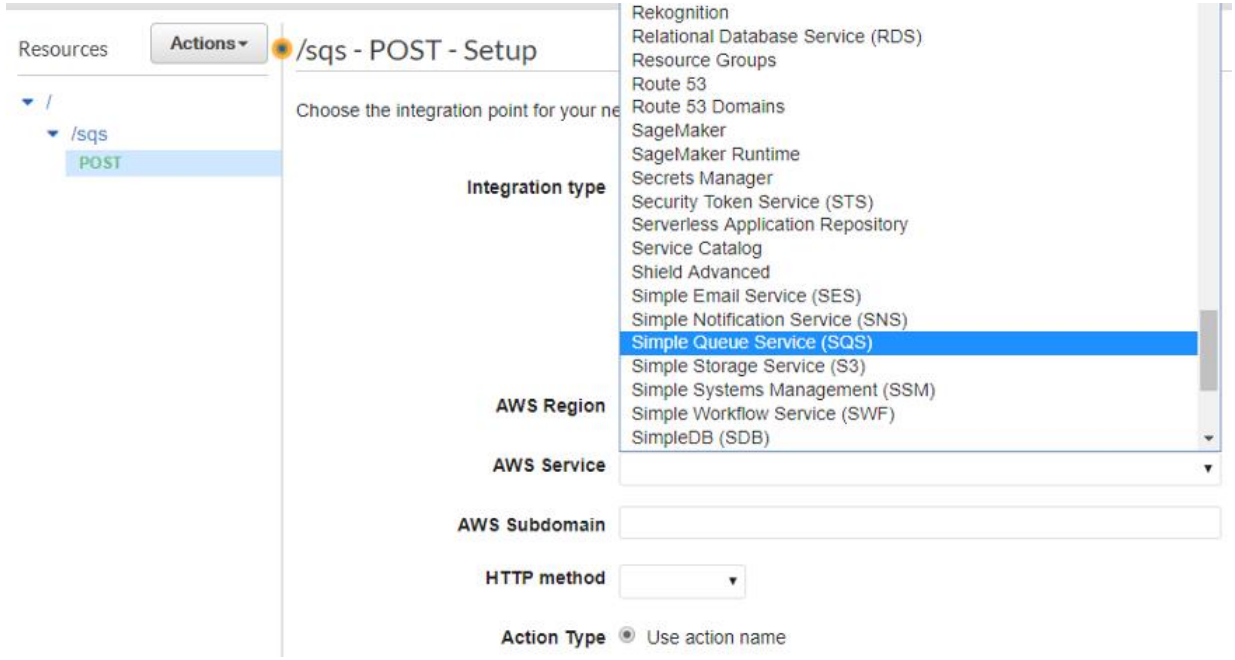
- Write fewer Lambda functions
  - use more direct service integrations in case the Lambda only calls the Service itself
- Write less Infrastructure as a Code
  - applies not only for Lambda but also to other services like AppSync
- Use (AWS) Services/Frameworks which provide meaningful abstractions
  - Amplify Framework or Serverless Framework Components

How to write less code with AWS Serverless services 3/3

- API Gateway Service Integration
- HTTP APIs Storage-First Service Integration
- Step Functions Service Integration
- Event Bridge Filtering and Routing
- Lambda Destinations
- AppSync and Direct Lambda Resolvers
- Amplify Framework
- Serverless Framework Components

API Gateway Service Integration

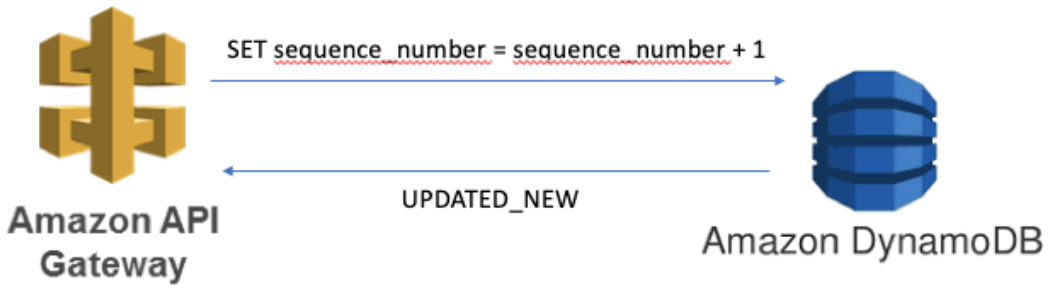


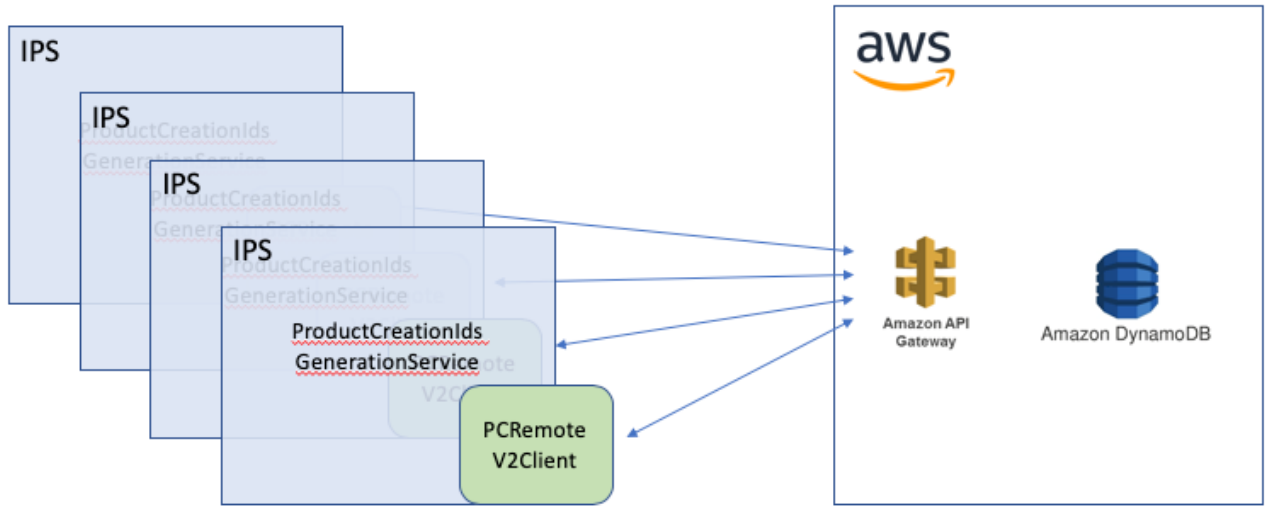


Sheen Brisals “Don’t wait for Functionless. Write less Functions instead”
https://medium.com/lego-engineering/dont-wait-for-functionless-write-less-functions-instead-8f2c331cd651

ip.labs

API Gateway Service Integration with Dynamo DB : Example ID generator 1/2

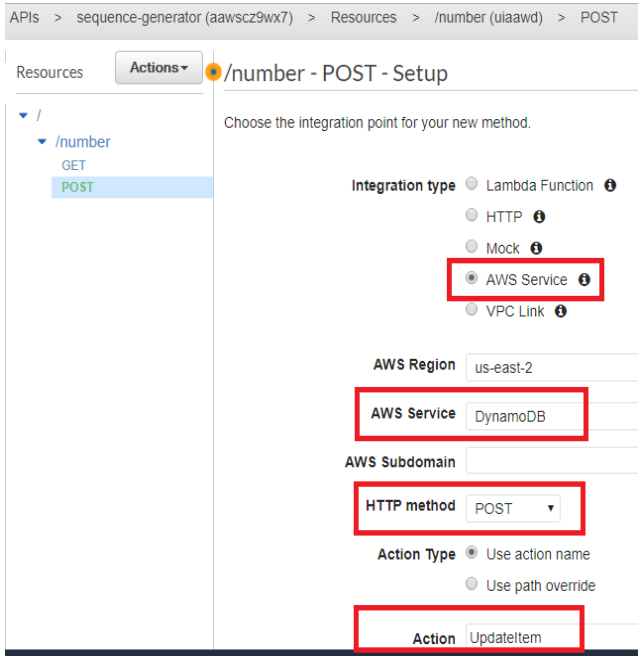


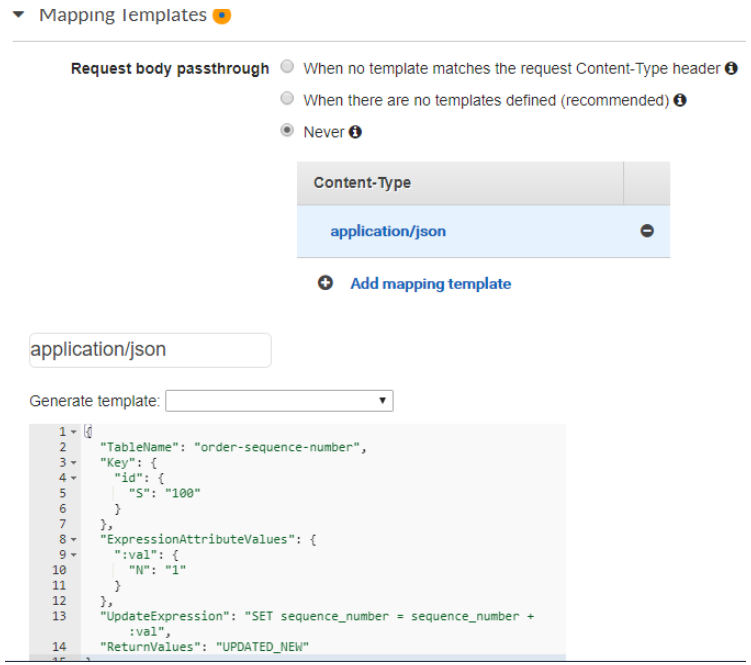


Sheen Brisals “Sequence Numbering in Serverless via API Gateway”
https://medium.com/lego-engineering/sequence-numbering-in-serverless-via-api-gateway-40e5f6c83e93
https://github.com/ToQoz/api-gateway-mapping-template

ip.labs

API Gateway Service Integration with Dynamo DB : Example ID generator



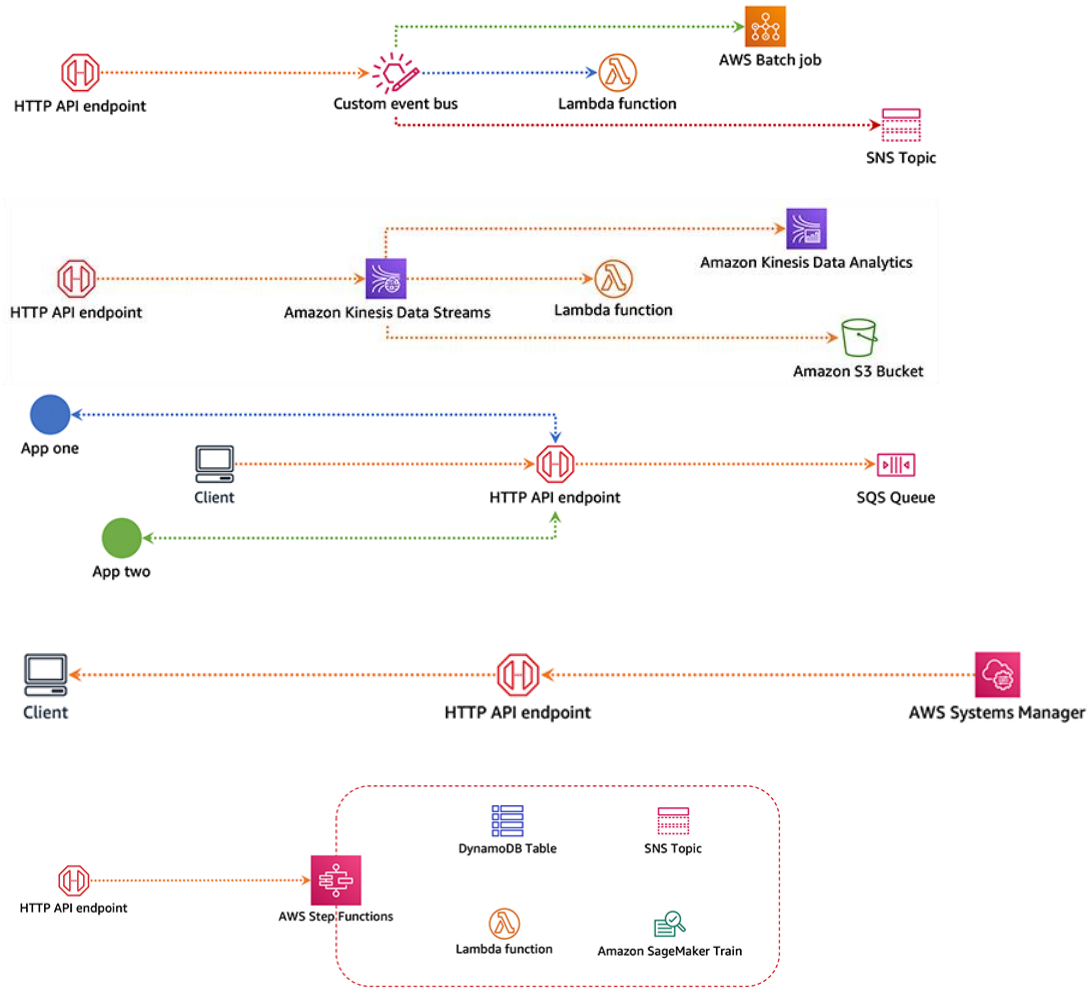


Sheen Brisals “Sequence Numbering in Serverless via API Gateway”
https://medium.com/lego-engineering/sequence-numbering-in-serverless-via-api-gateway-40e5f6c83e93
https://github.com/ToQoz/api-gateway-mapping-template

ip.labs

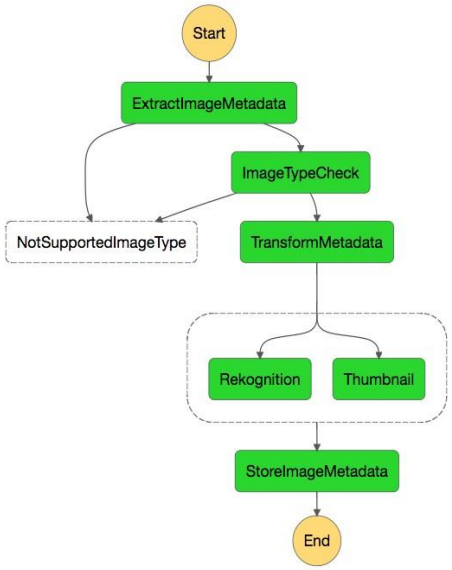
HTTP APIs Storage-First Service Integration

- Event Bridge
- Kinesis Data Streams
- SQS
- AppConfig
- Step Functions



Eric Johnson “https://aws.amazon.com/de/blogs/compute/building-storage-first-applications-with-http-apis-service-integrations/”

Step Functions Service Integration

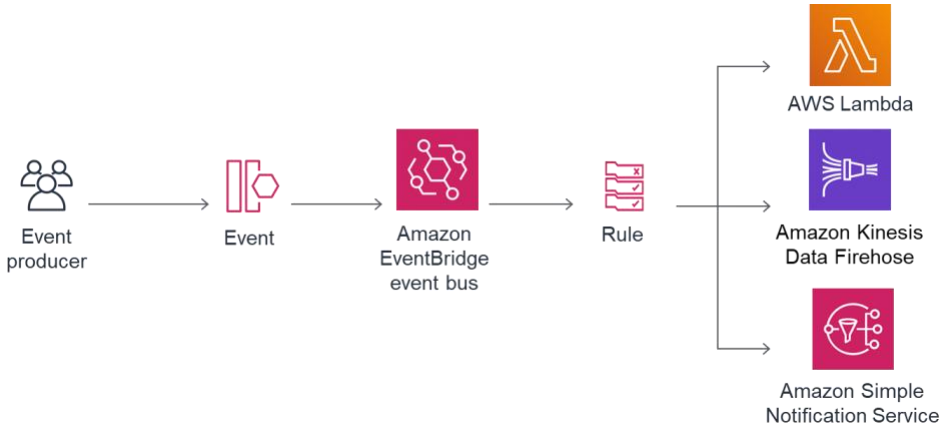


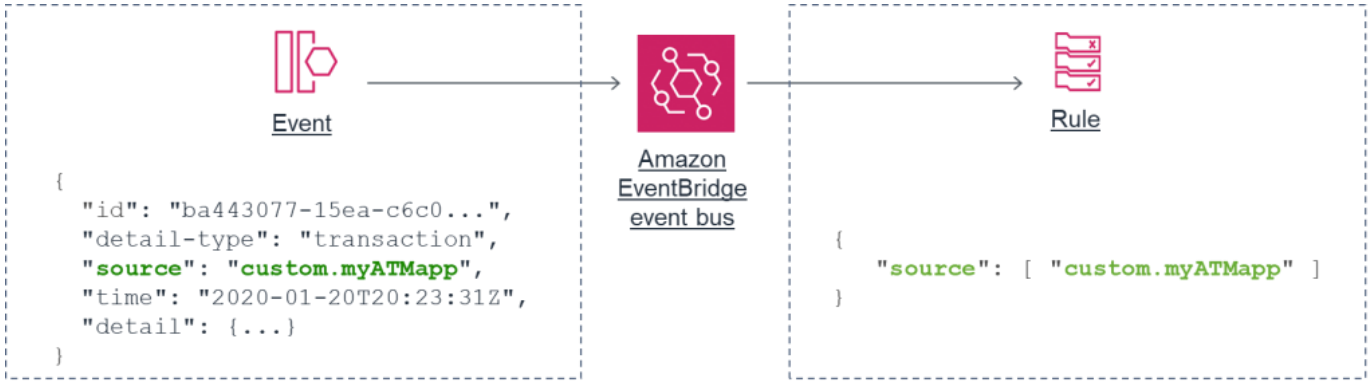
Standard Workflows | Express Workflows

Supported Service Integrations

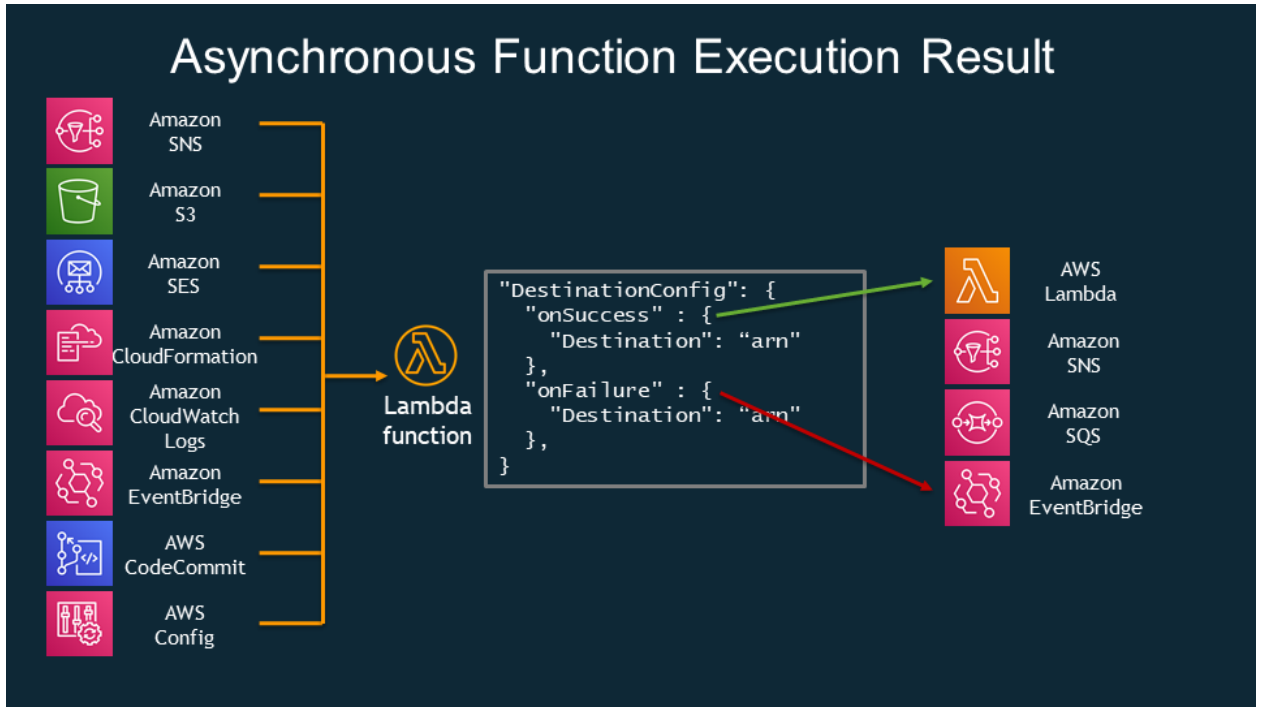
Service	Request Response	Run a Job (.sync)	Wait for Callback (.waitForTaskToken)
Lambda	✓		✓
AWS Batch	✓	✓	
DynamoDB	✓		
Amazon ECS/AWS Fargate	✓	✓	✓
Amazon SNS	✓		✓
Amazon SQS	✓		✓
AWS Glue	✓	✓	
SageMaker	✓	✓	
Amazon EMR	✓	✓	
AWS CodeBuild	✓	✓	
AWS Step Functions	✓	✓	✓

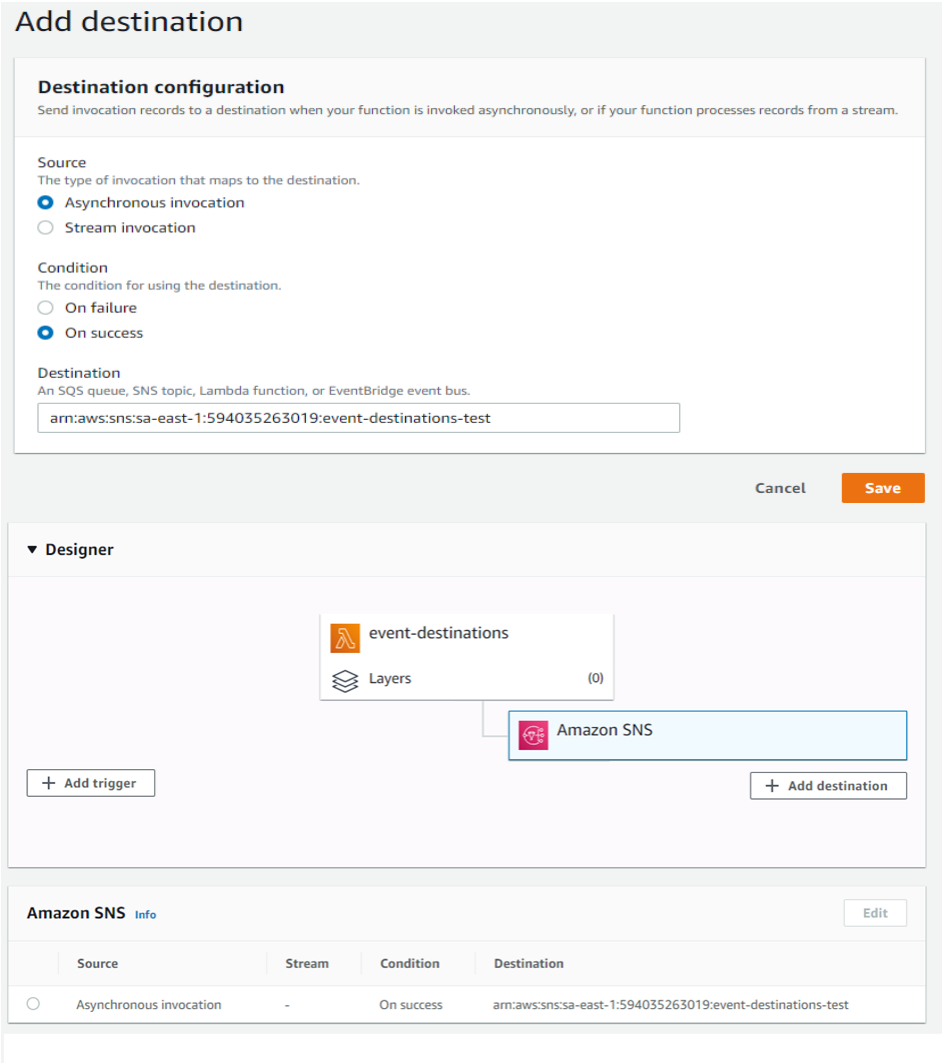
Event Bridge Filtering and Routing





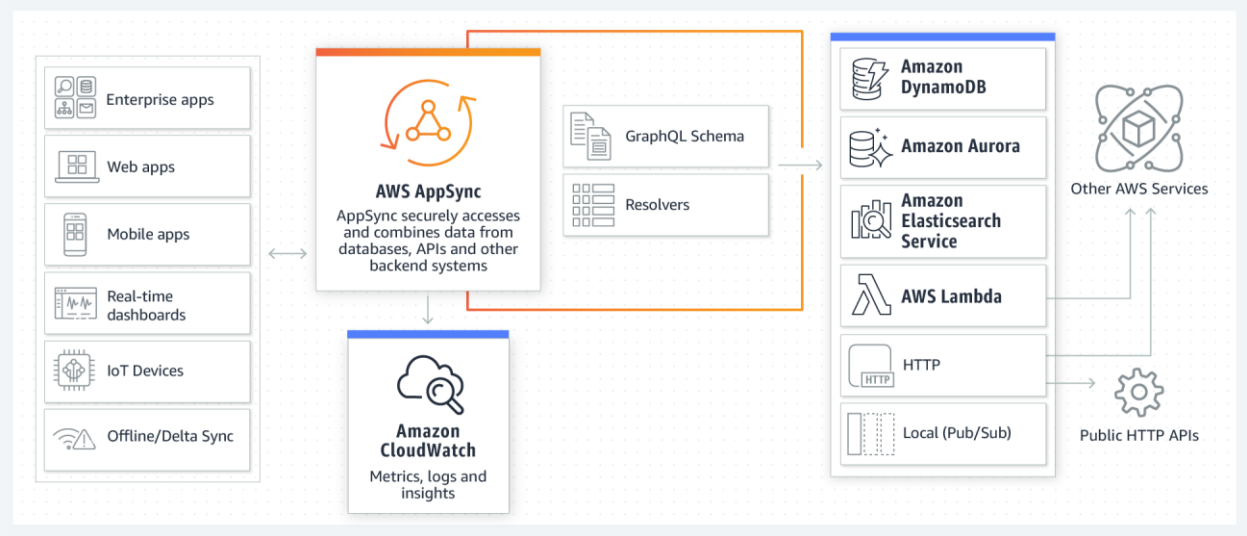
Lambda Destinations

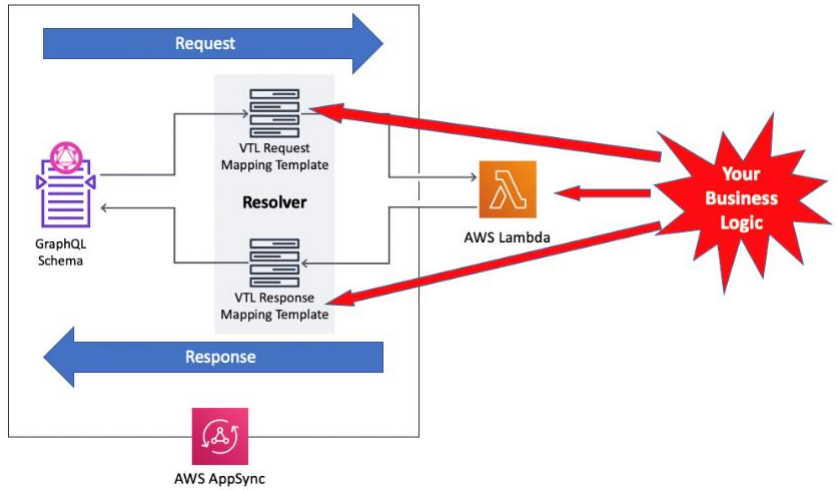


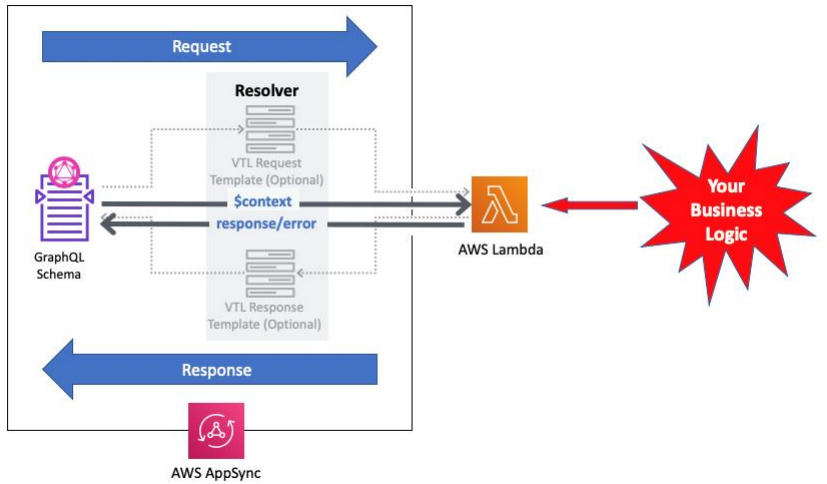


"Introducing AWS Lambda Destinations" https://aws.amazon.com/de/blogs/compute/introducing-aws-lambda-destinations/
https://www.trek10.com/blog/lambda-destinations-what-we-learned-the-hard-way

AppSync and Direct Lambda Resolvers



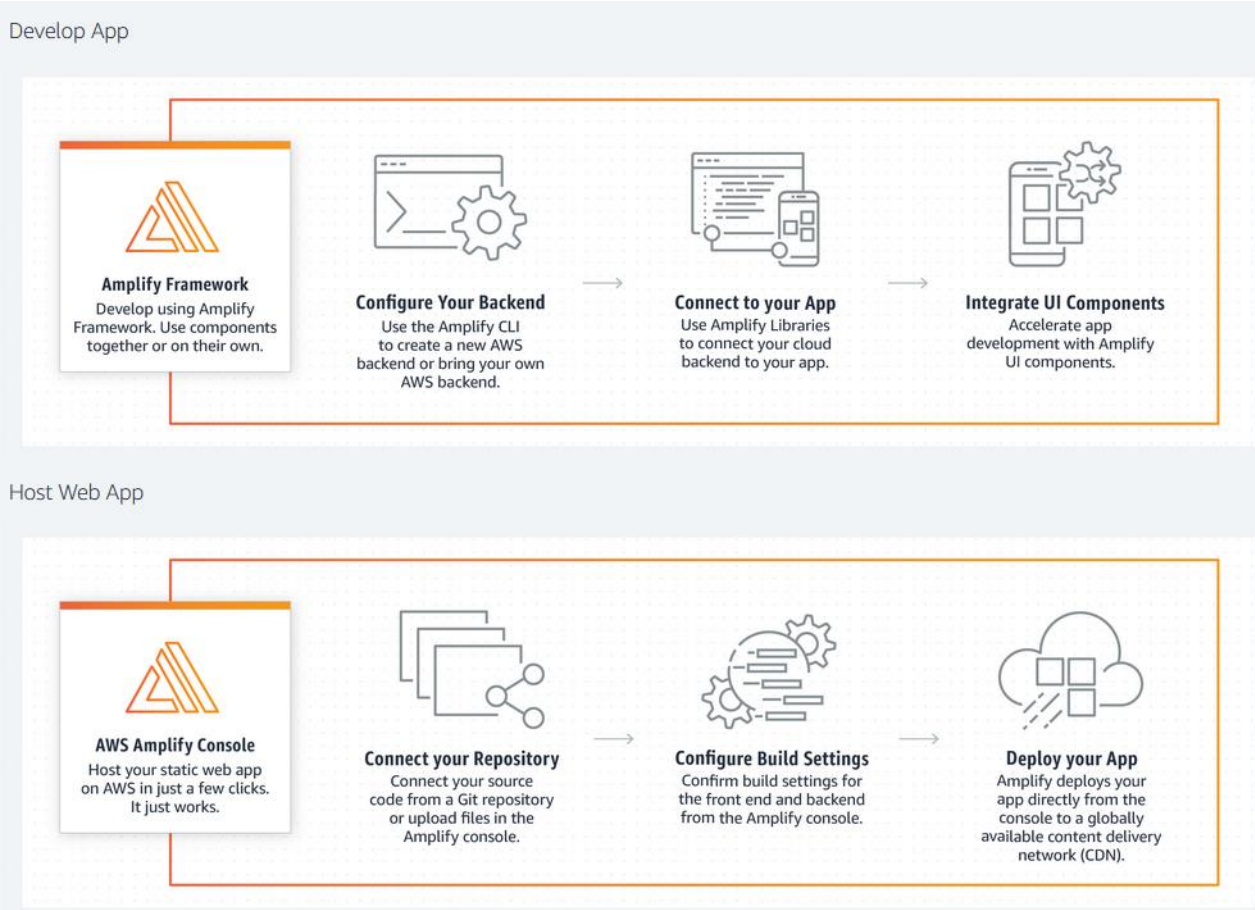




https://aws.amazon.com/de/appsync
“Introducing Direct Lambda Resolvers: AWS AppSync GraphQL APIs without VTL” https://aws.amazon.com/de/blogs/mobile/appsync-direct-lambda/

ip.labs

Amplify Framework



https://aws.amazon.com/amplify/?nc1=h_ls

ip.labs

Serverless Framework Components

Serverless Components is now Generally Available. Click Here for the old Beta Version

English | 简体中文

Serverless Components are simple abstractions that enable developers to deploy serverless applications and use-cases easily, via the Serverless Framework.

- **Ease** - Deploy entire serverless applications/use-cases via Components, without being a cloud expert.
- **Instant Deployments** - Components deploy in 2-4 seconds, making rapid development on the cloud possible.
- **Streaming Logs** - Many Components stream logs from your app to your console in real-time, for fast debugging.
- **Automatic Metrics** - Many Components auto-set-up metrics upon deployment.
- **Build Your Own** - Components are easy to build.
- **Registry** - Share your Components with you, your team, and the world, via the Serverless Registry.

https://www.serverless.com/blog/what-are-serverless-components-how-use

Using Serverless ecosystem will

with the right engineering practices in place will significantly reduce

- extraneous and germane cognitive load
- the amount of code written
- amount of dependencies (no execution runtime and web application server to take care of)

Serverless with the focus on you core domains will enable

- iterative development mind set
- experimentation culture
- focus on business value innovation and faster time to market
- evolutionary architectures

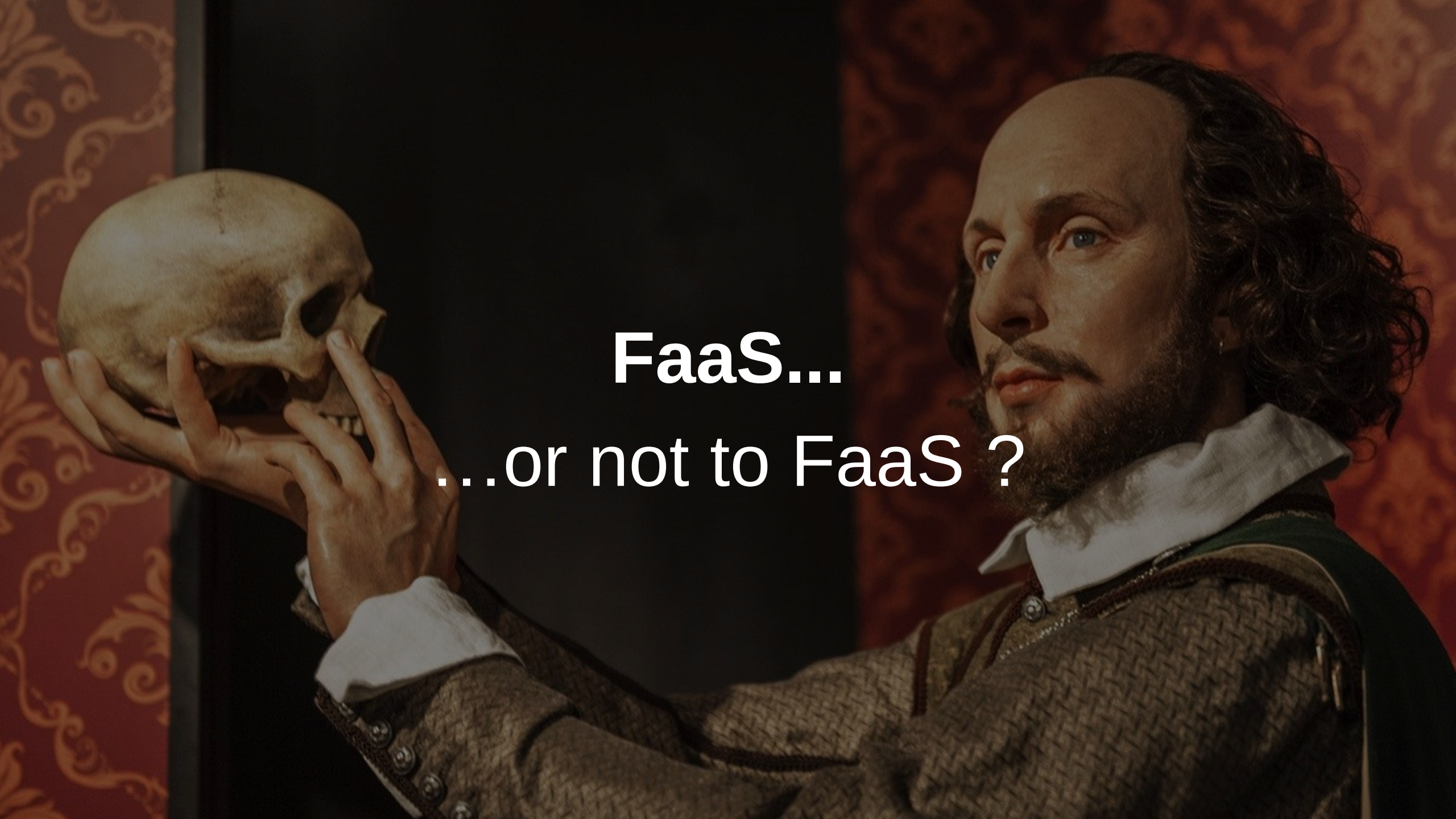

FaaS...
…or not to FaaS ?

Image: https://stock.adobe.com

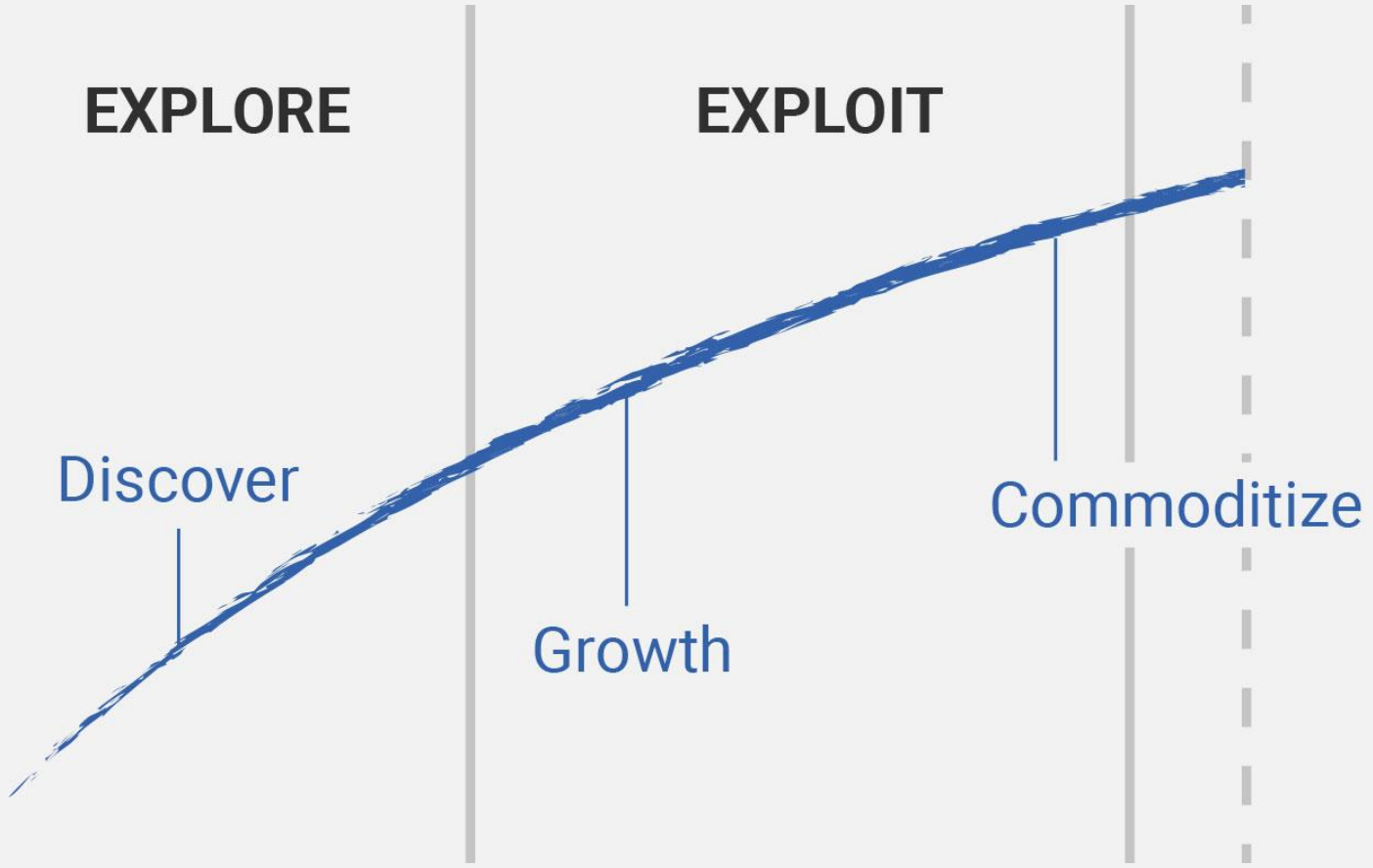
Decision Checklist: understand your…

1. Application lifecycle
2. Workloads
3. Programming Model
4. Platform limitations
5. Cost at scale
6. Organizational environment
7. Platform and tooling maturity

Decision Checklist: understand your…

1. Application lifecycle
2. Workloads
3. Programming Model
4. Platform limitations
5. Cost at scale
6. Organizational environment
7. Platform and tooling maturity

Understand Application lifecycle



Explore phase

- Quickly validate hypotheses
- Rapidly experiment
- Run experiments as cheaply as possible

⬇

Serverless is a perfect fit

Exploit phase

- Built something that does provide customer value
- Build it on scale
- Build a profitable product around it

⬇

partly serverless and partly not serverless architecture

Image: Robert Scoble via Flickr

ip.labs

Application lifecycle

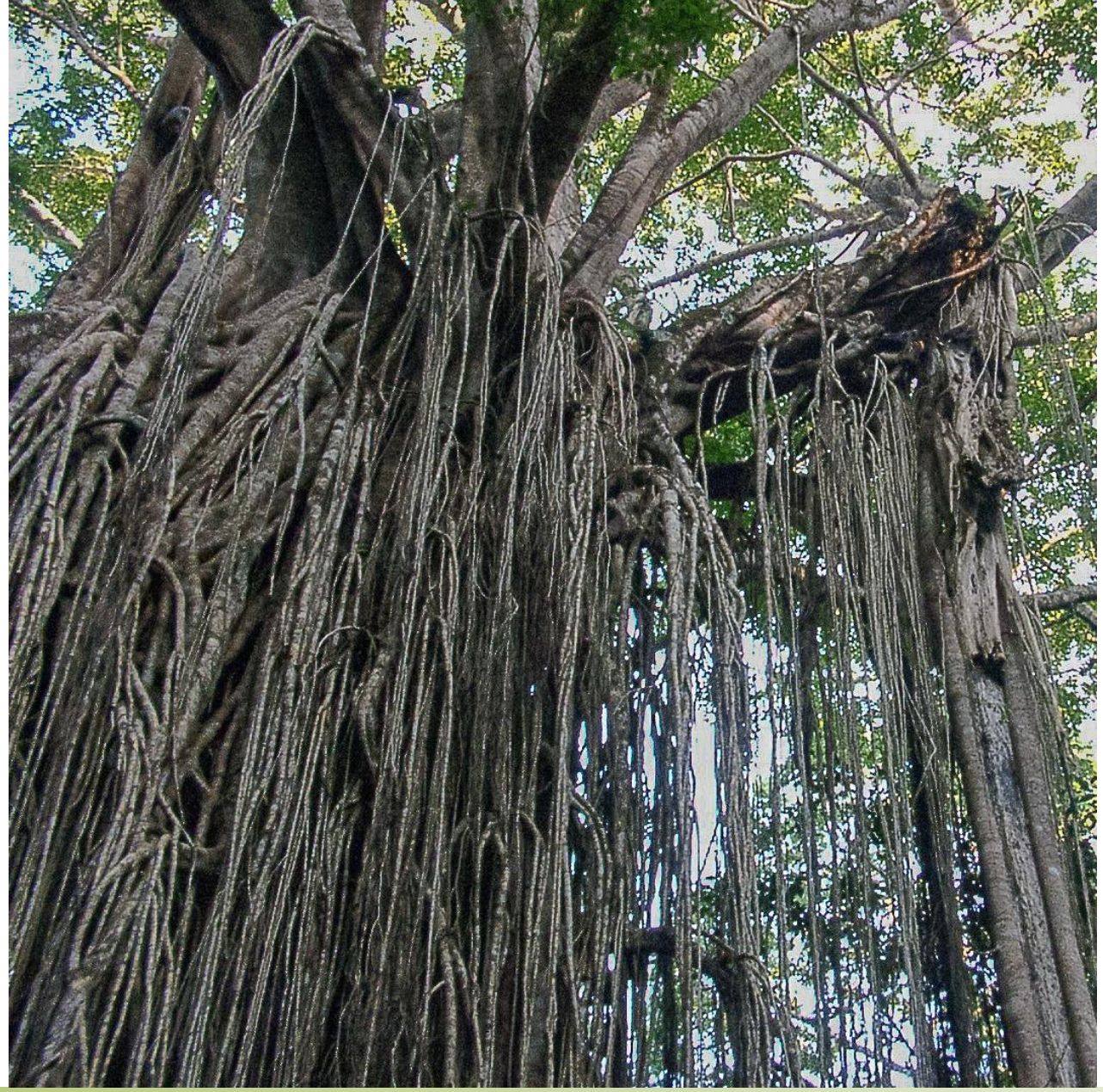
- How much of my stack should I own to be able to deliver business value?
- Outsource SLA, regulatory compliance, price, and roadmap to my service provider?

Existing applications

- You can’t magically move that all off to service providers
- You can try to modernize parts of them

Strangler Pattern

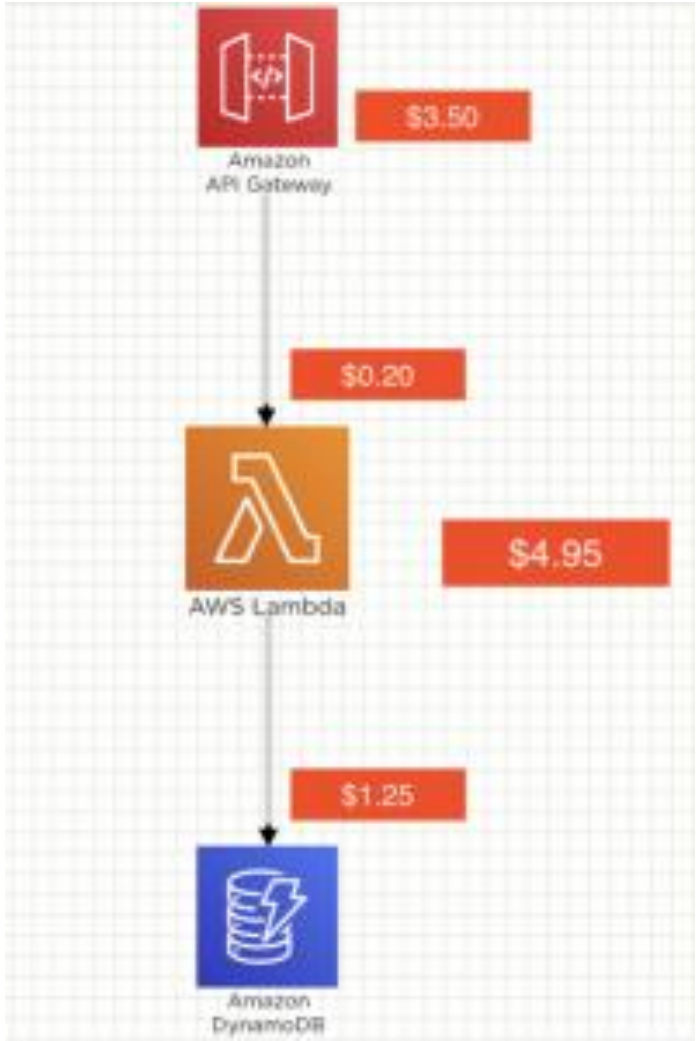
- Add a proxy (API Gateway or Application Loadbalancer), which sits between the legacy application and the user
- Add new services and link it to the proxy



FinDev Concept

Activity-based costing on a digital operation-by-operation basis

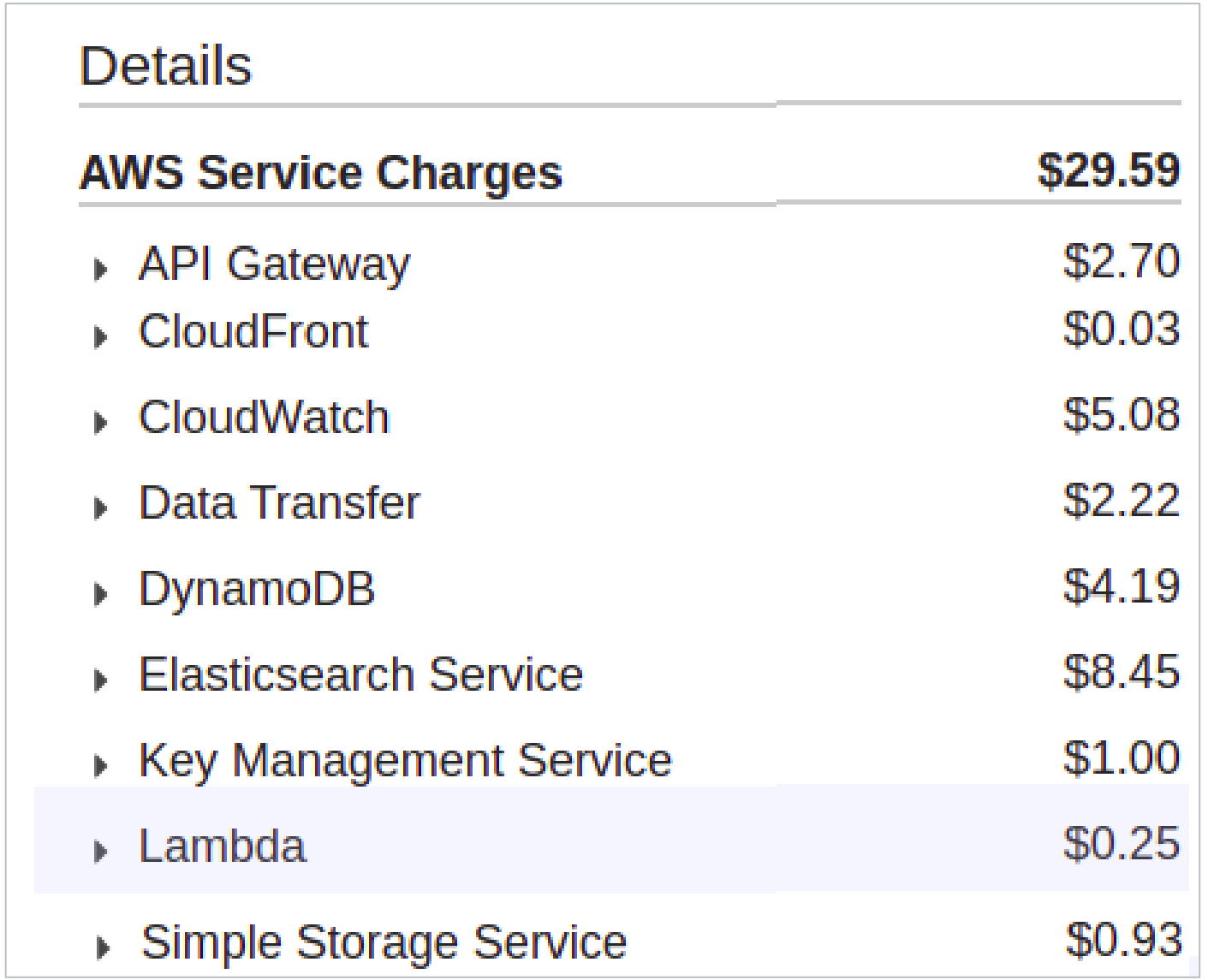
- Figure out features which deliver business value comparing to their cost



1. Application lifecycle
2. Workloads
3. Programming Model
4. Platform limitations
5. Cost at scale
6. Organizational environment
7. Platform and tooling maturity

The reality is…

Lambda is often just a small percentage of your total cost



API Gateway

$ 3.50

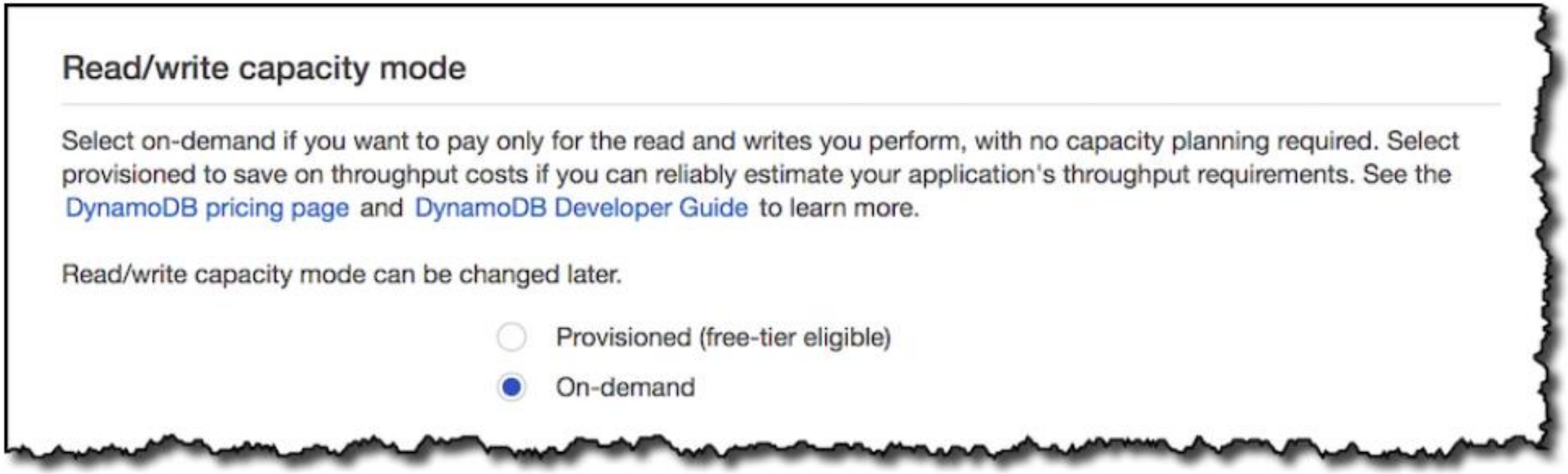
Per million API calls

HTTP APIs in Beta

70% cheaper as API Gateway

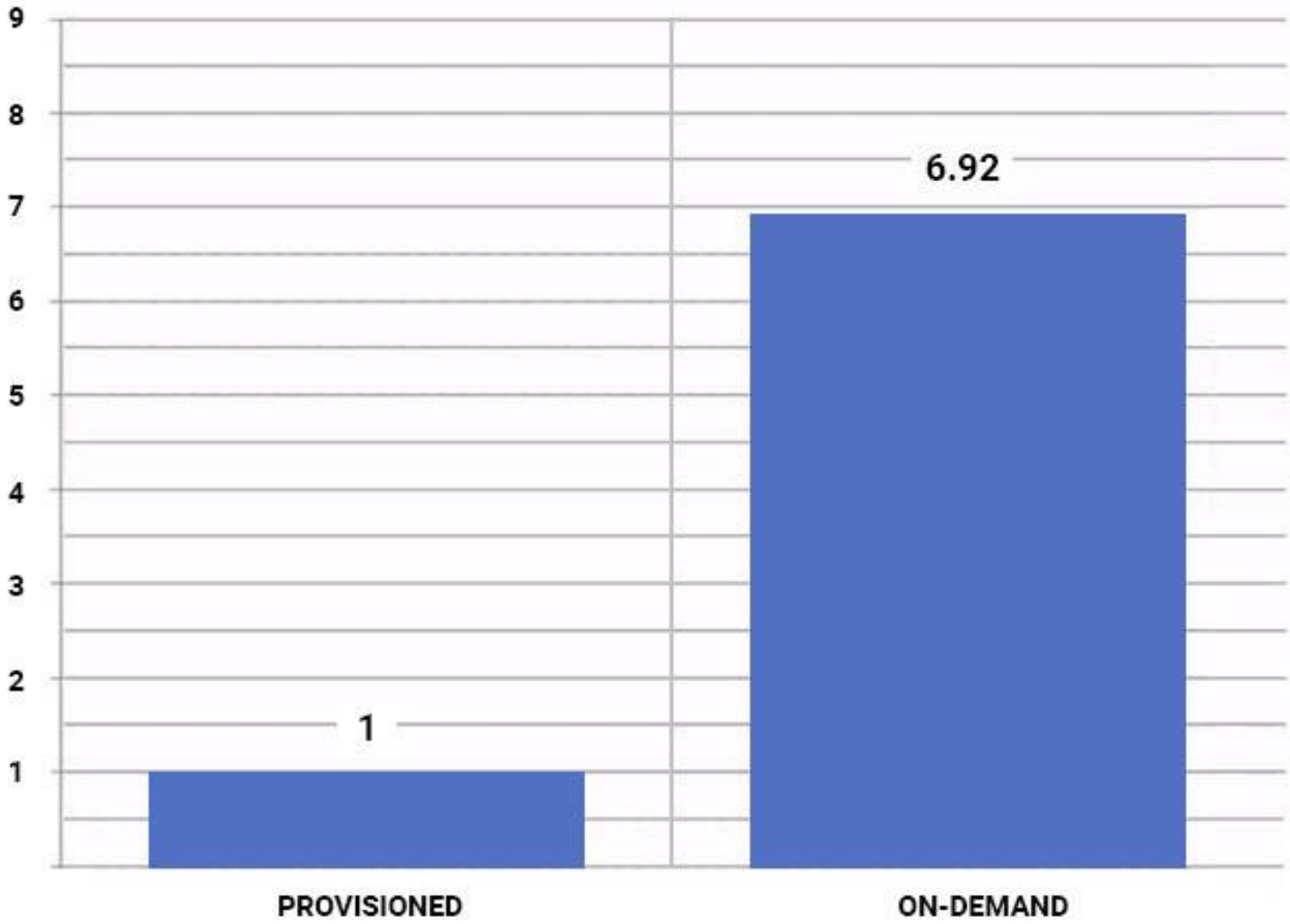
- Fewer configuration options
- Well-suited for most use-cases

DynamoDB On-Demand



Provisioned vs On-Demand

- Use On-Demand for spiky workloads
- Use Provisioned for constantly high workload

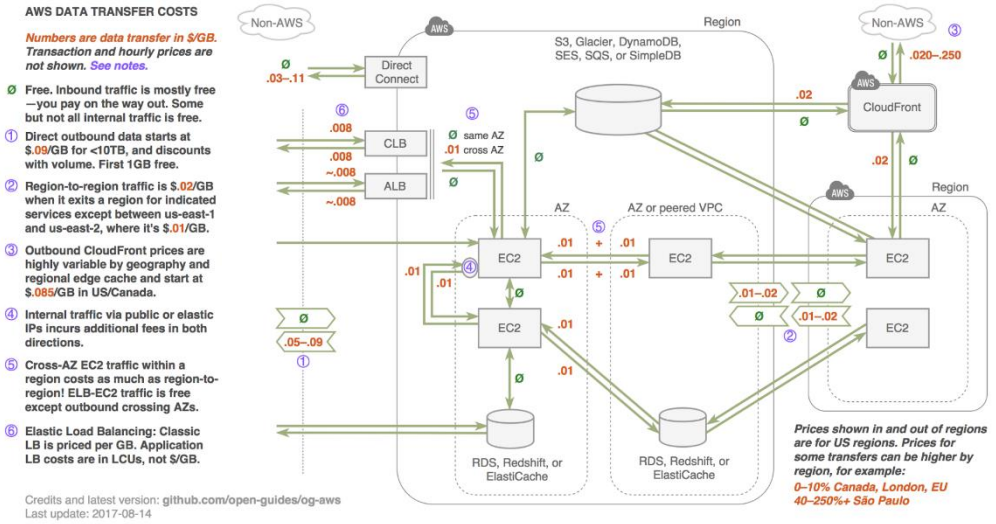


Understand your cost at scale

- Lambda
- API Gateway
- Dynamo DB capacity choices
- Logging costs
- Monitoring costs

Understand your cost at scale

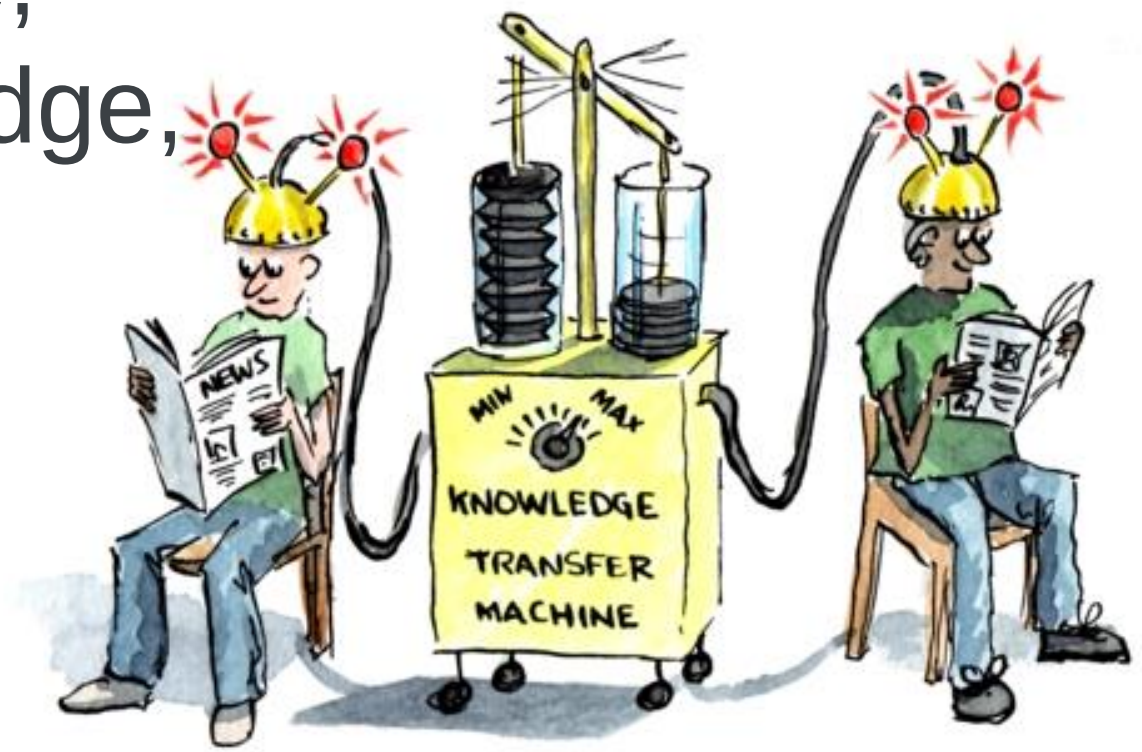
- Data transfer costs
- Step functions
- Remote API calls
- 3rd party Services price models



1. Application lifecycle
2. Workloads
3. Programming Model
4. Platform limitations
5. Cost at scale
6. Organizational environment
7. Platform and tooling maturity

Understand organizational environment (structure, reponsibilities, knowledge, culture)

- Do you already embrace DevOps best practices?



Charity Majors: „The Future of Ops careers“

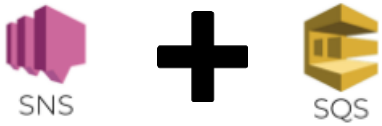
- Advocates for the internal observability team (even if you pay for SaaS observability solution and you use Serverless-first approach)
  - team should write libraries, generate examples, and drive standardization; ushering in consistency, predictability, and usability
  - team should partner with internal teams to evaluate use cases. They might also write glue code and helper modules to connect different data sources and create cohesive visualizations
  - team becomes an integration point between your organization and the outsourced work

Tom McLaughlin Talk: „What do we do when the server goes away?“

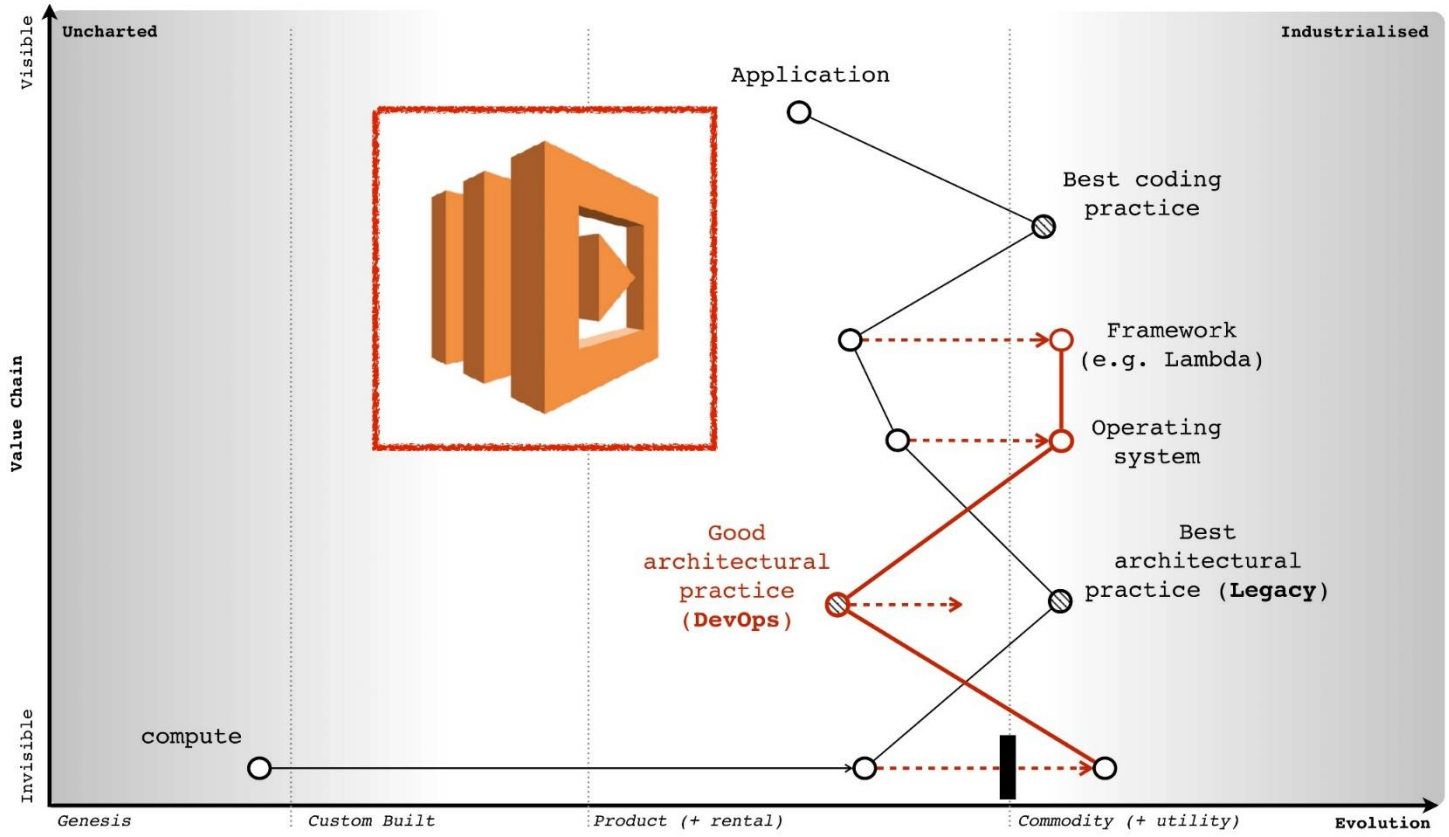
- Observability (Monitoring, Logging, Tracing & Alerting)
- Chaos Engineering & Game Days
- Infrastructure as Code & Testing
- Help understand constraints

of AWS services & choose the right one

Help understand constraints of AWS services & choose the right one. Example Event Sources:



Wardley Map


Uncharted
Industrialised
Visible
Value Chain
Invisible
Application
Best coding practice
Framework (e.g. Lambda)
Operating system
Good architectural practice (DevOps)
Best architectural practice (Legacy)
compute
Genesis
Custom Built
Product (+ rental)
Commodity (+ utility)
Evolution

Co-evolution of practices with Serverless 1/2

- True DevOps (even DevSecOps)
- FinDev responsibilities in the teams
- Complete infrastructure automation
- Chaos Engineering

Dev Ops

Type 2 suitability: organisations with a single main web-based product or service.
Potential effectiveness: HIGH

Type 2: Fully Shared Ops Responsibilities

Where operations people have been integrated in product development teams, we see a Type 2 topology. There is so little separation between Dev and Ops that all people are highly focused on a shared purpose; this is arguable a form of **Type 1 (Dev and Ops Collaboration)**, but it has some special features.

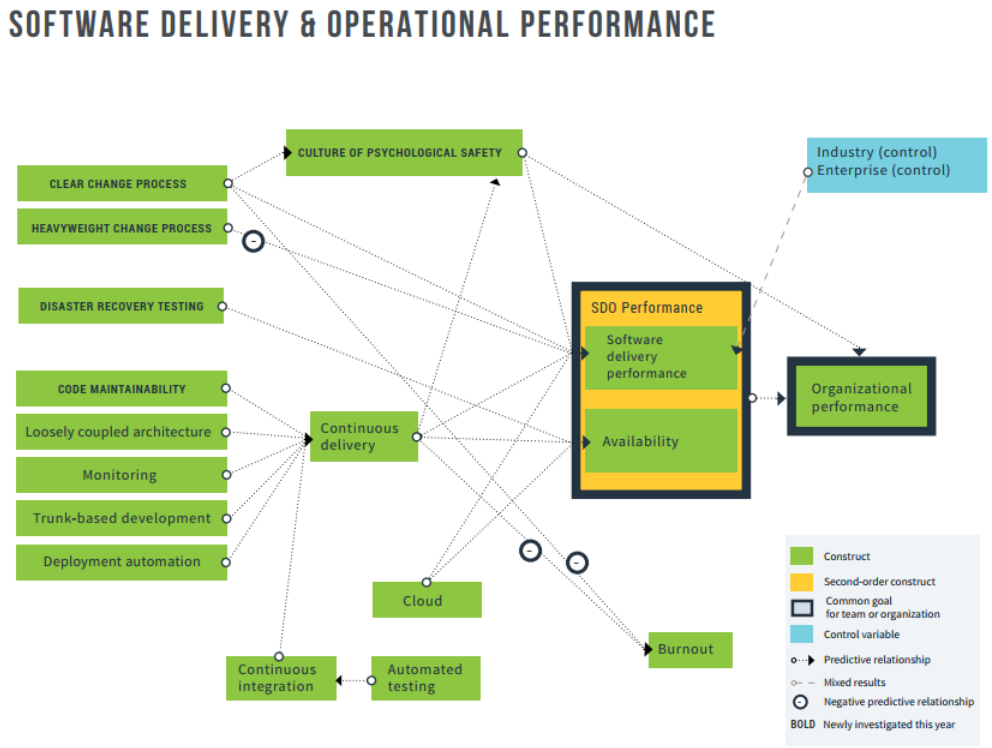
Organisations such as Netflix and Facebook with effectively a single web-based product have achieved this Type 2 topology, but I think it's probably not hugely applicable outside a narrow product focus, because the budgetary constraints and context-switching typically present in an organisation with multiple product streams will probably force Dev and Ops further apart (say, back to a **Type 1 model**). This topology might also be called 'NoOps', as there is no distinct or visible Operations team (although the Netflix NoOps might also be **Type 3 (Ops as IaaS)**).

Sheen Brisals "Why the 'WHY' matters more than the 'WHAT' in Serverless!"
https://medium.com/lego-engineering/why-the-why-matters-more-than-the-what-in-serverless-2ef56c397962
DevOps Topologies: https://web.devopstopologies.com/

Co-evolution of practices with Serverless 2/2

- Each team or even developer can get its own (AWS) test account
- No local testing environment or only for quick functional tests
- Testing in production

Invest in Software Delivery and Operational Performance Excellence



See DORA State of DevOps 2018-2019 Reports

ip.labs

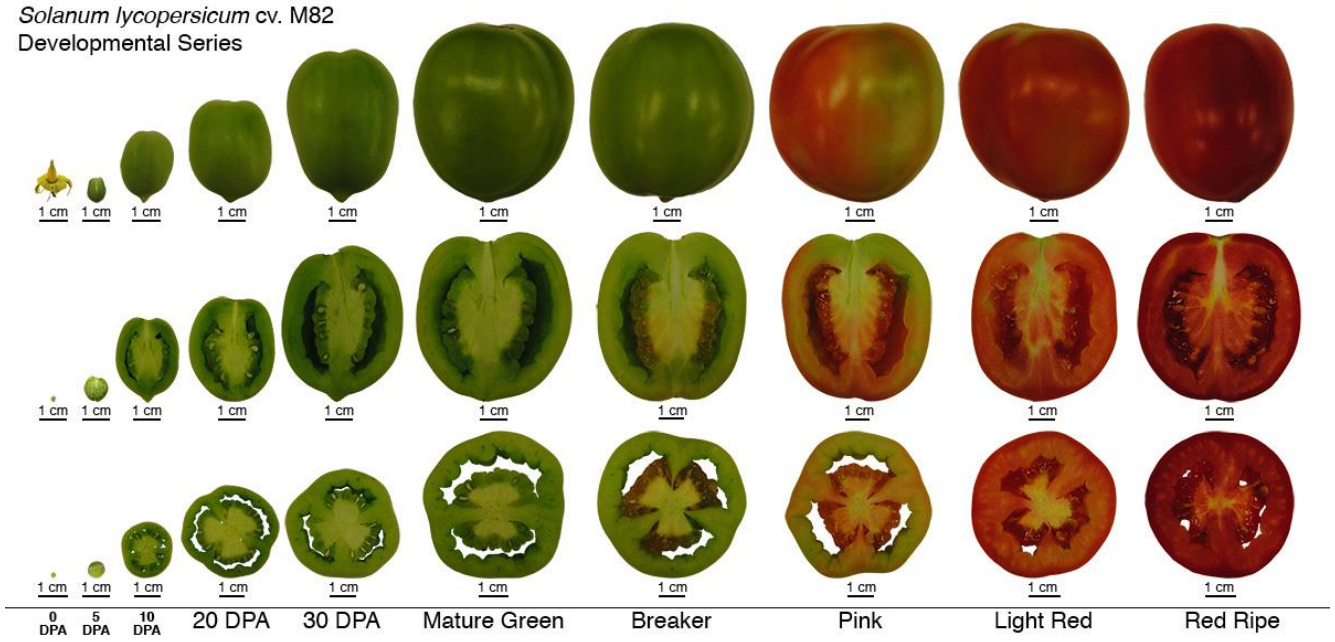
1. Application lifecycle
2. Workloads
3. Programming Model
4. Platform limitations
5. Cost at scale
6. Organizational environment
7. Platform and tooling maturity

Serverless platform and tooling maturity

- Infrastructure-as-a-Code solutions maturity (e.g. Cloud Formation, CDK, Terraform)
- Development environment & framework maturity (e.g. AWS SAM, AWS Amplify, Serverless Framework)
- Integration with 3rd party SaaS

Serverless platform and tooling maturity

- CI/CD
- Observability (Logging, Monitoring, Tracing)
- Alerting
- Security



Thank You!