Cloud Trends

Principles, Evolution, and Chaos...

Adrian Cockcroft @adrianco
VP Cloud Architecture Strategy

aws

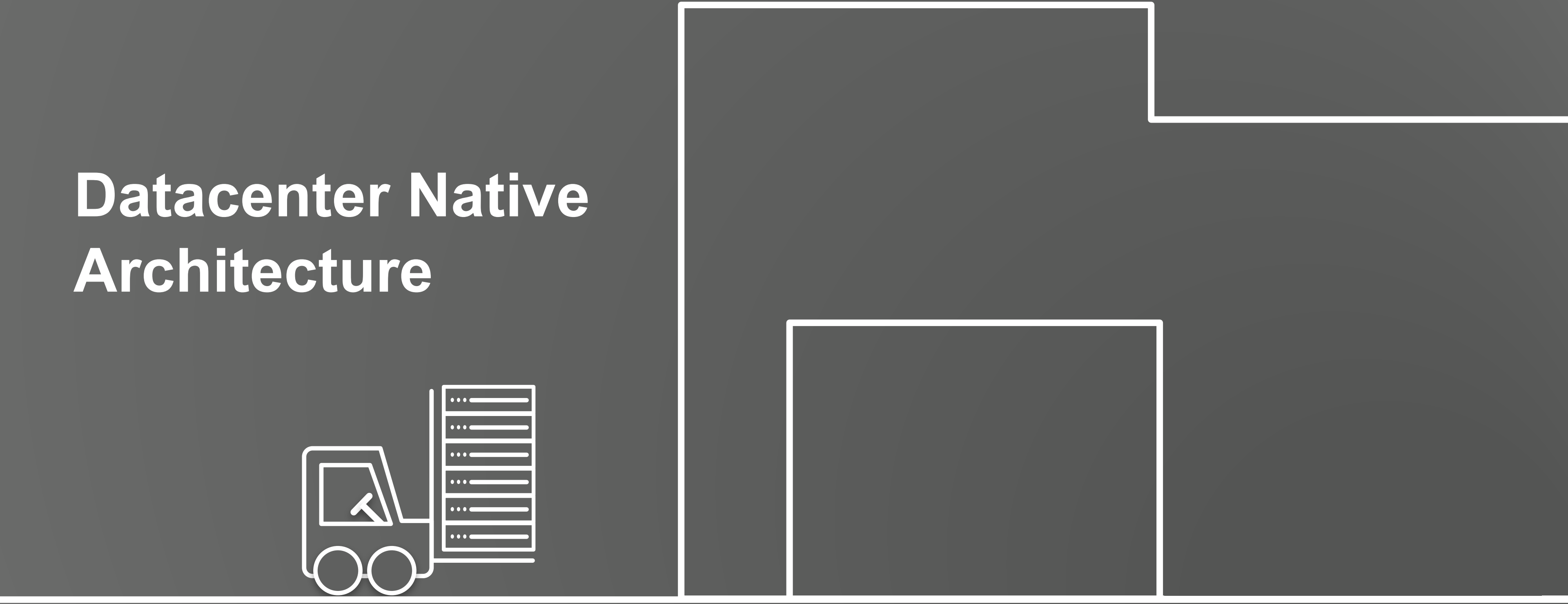
Cloud Native Architecture

Principles and Practice

Adrian Cockcroft

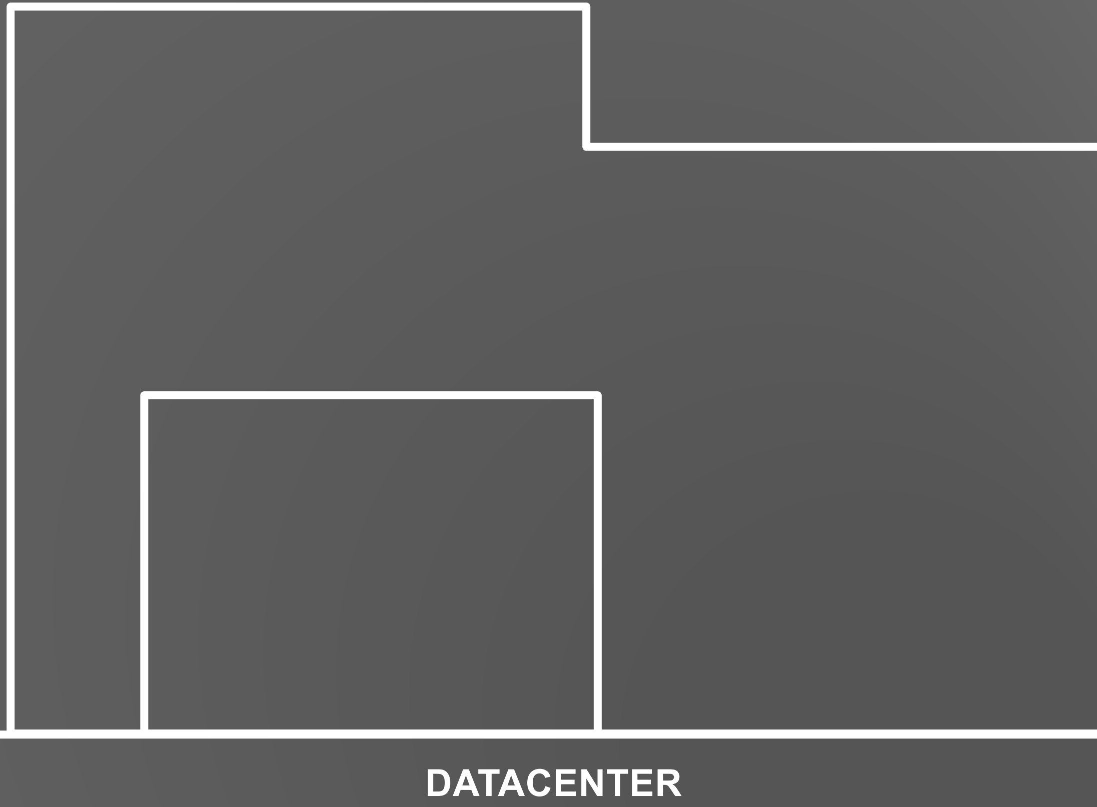
What is Cloud Native?

Datacenter Native Architecture



Datacenter Native Architecture e

Lives for years

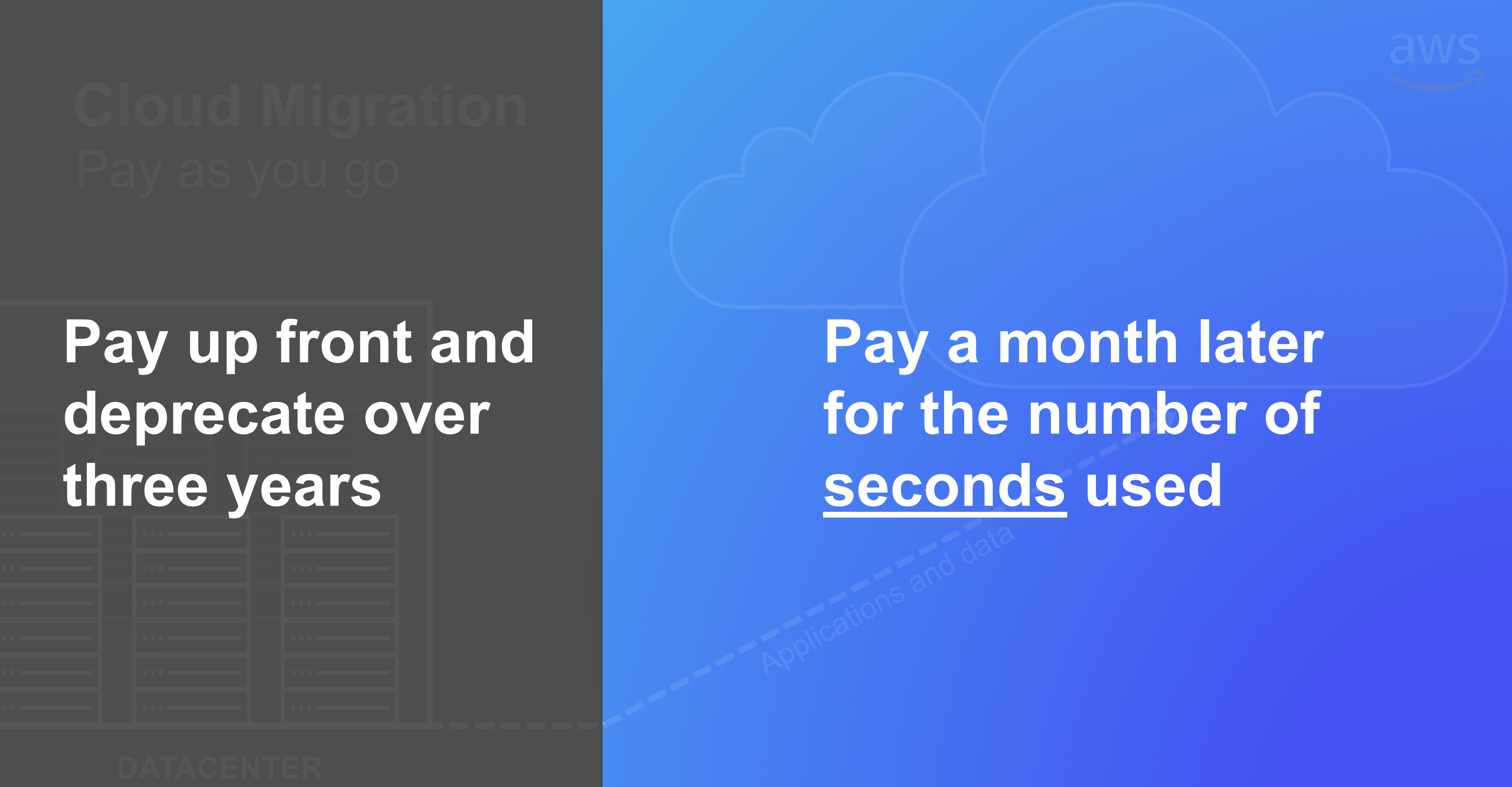


Cloud Migration

Pay as you go

Pay up front and deprecate over three years

Pay a month later for the number of seconds used

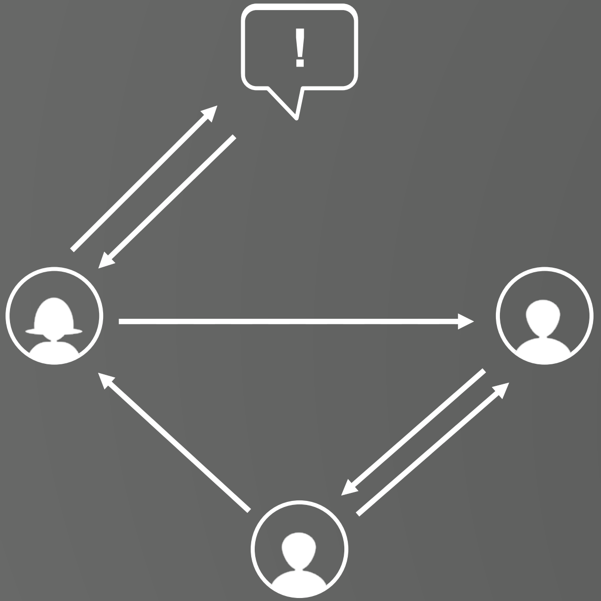


Cloud Native Principle

Pay for what you used last month.

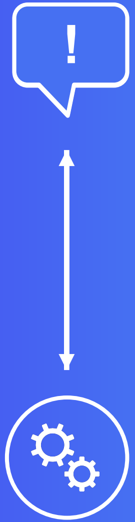
Not what you guess you will need next year.

File tickets and wait for every step

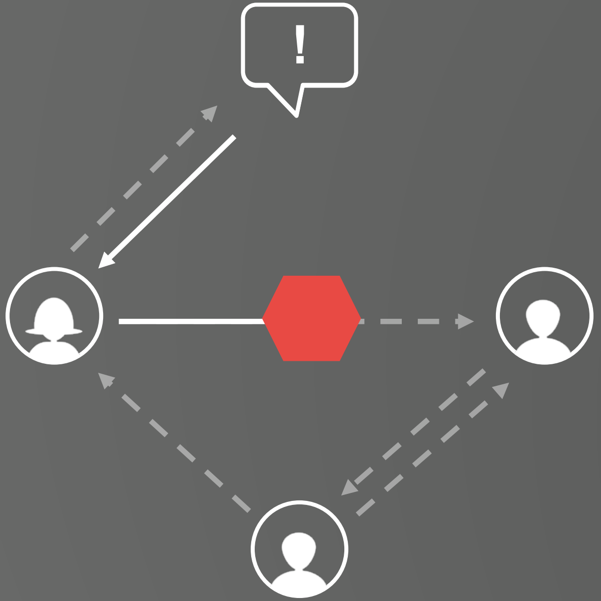


VS

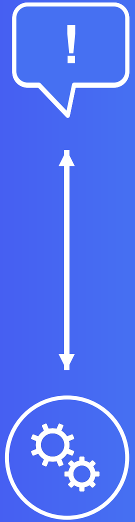
Self service, on-demand, no delays



File tickets and wait for every step

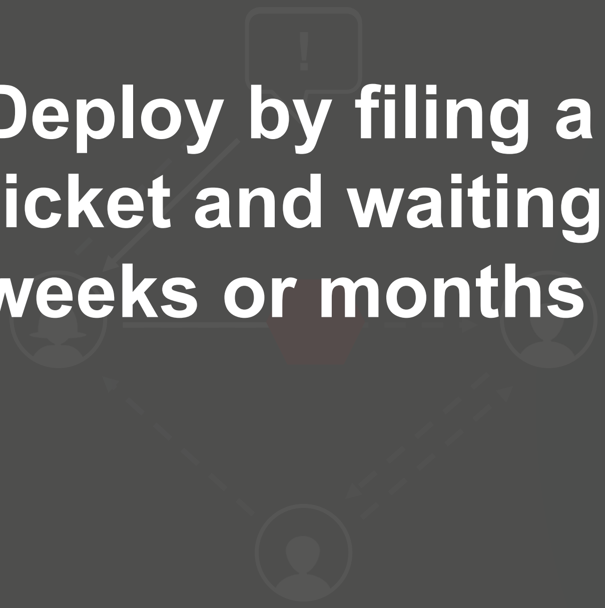


Self service, on-demand, no delays



File tickets and wait for every step

Self service, on-demand, no delays



vs

Deploy by filing a ticket and waiting weeks or months

Deploy by making an API call self service within minutes

Cloud Native Principle

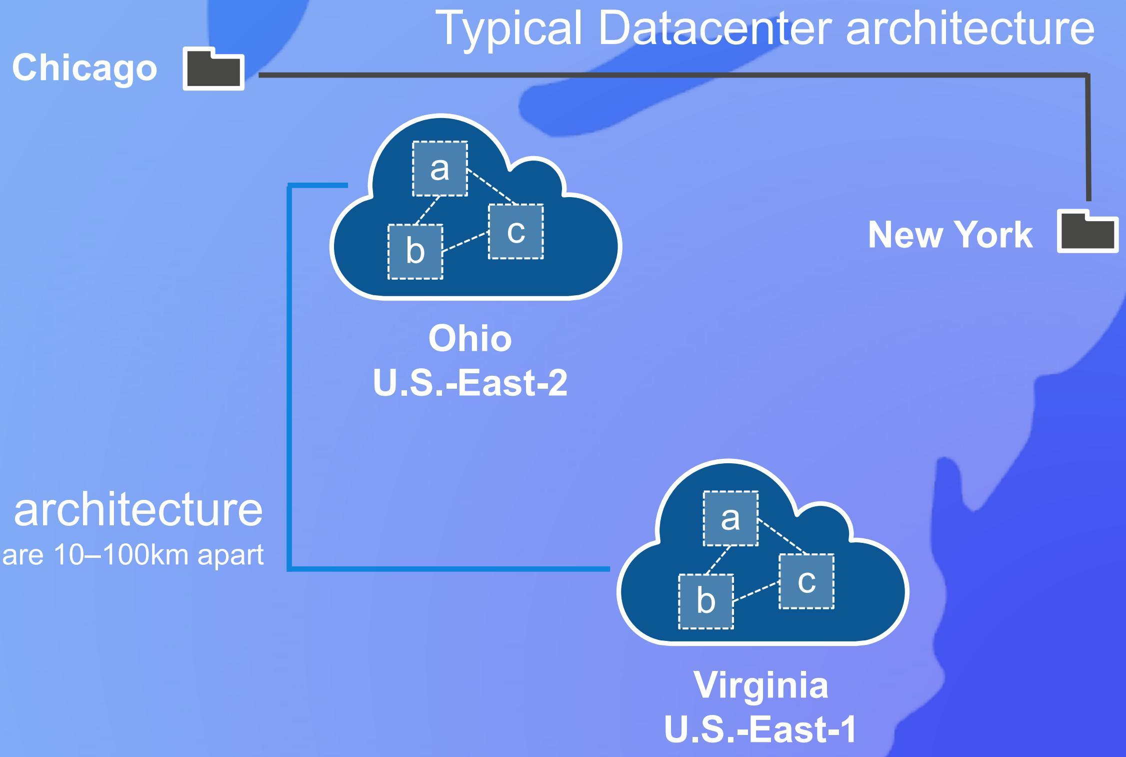
Self service, API driven, automated.

Move from request tickets at every step to a tracking ticket that records what happened.

Cloud Native Principle

Instant globally distributed deployments and data by default.

Regions and Zones

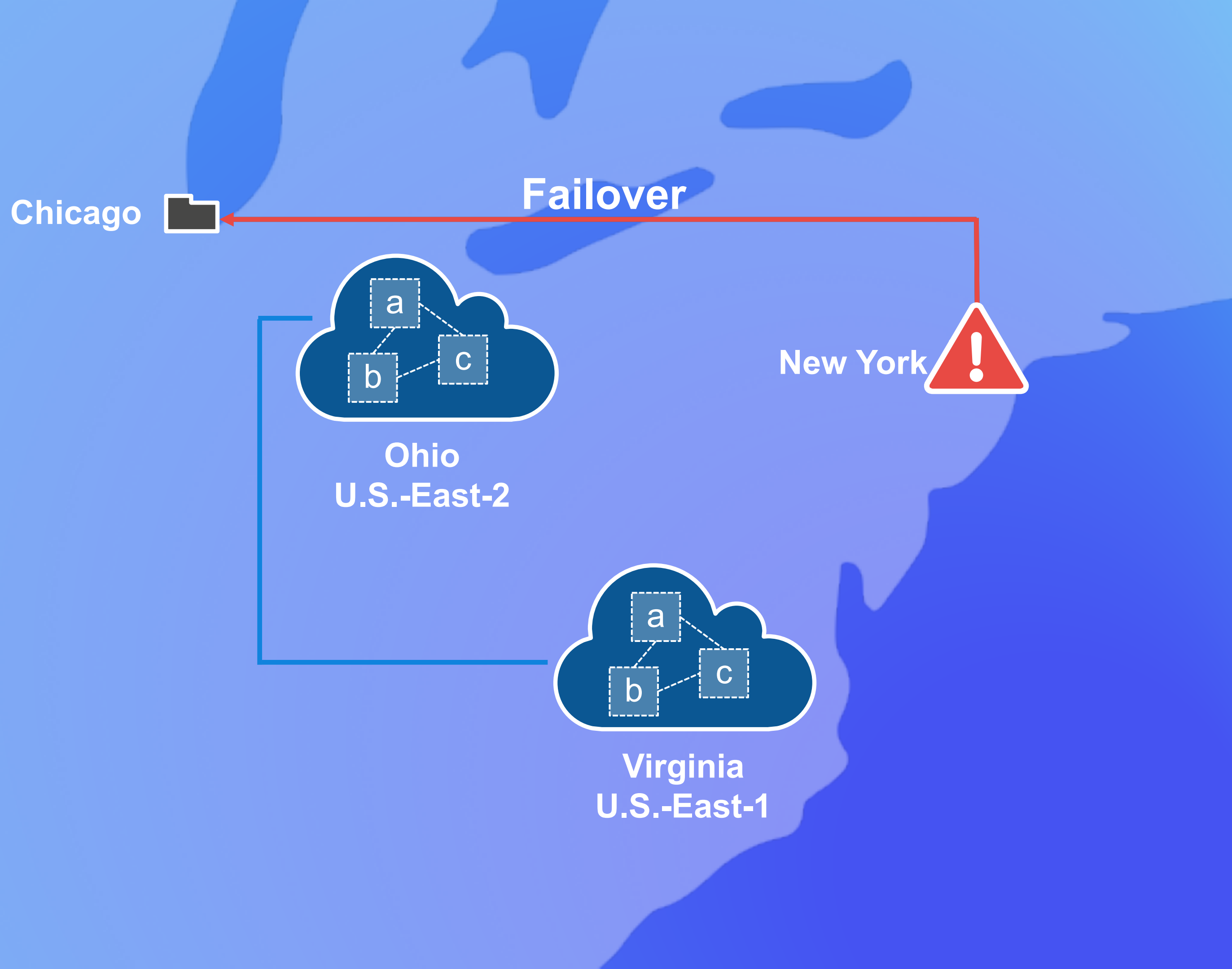


Typical cloud architecture

Zones a, b, and c are 10–100km apart

Regions and Zones

Hurricane Sandy

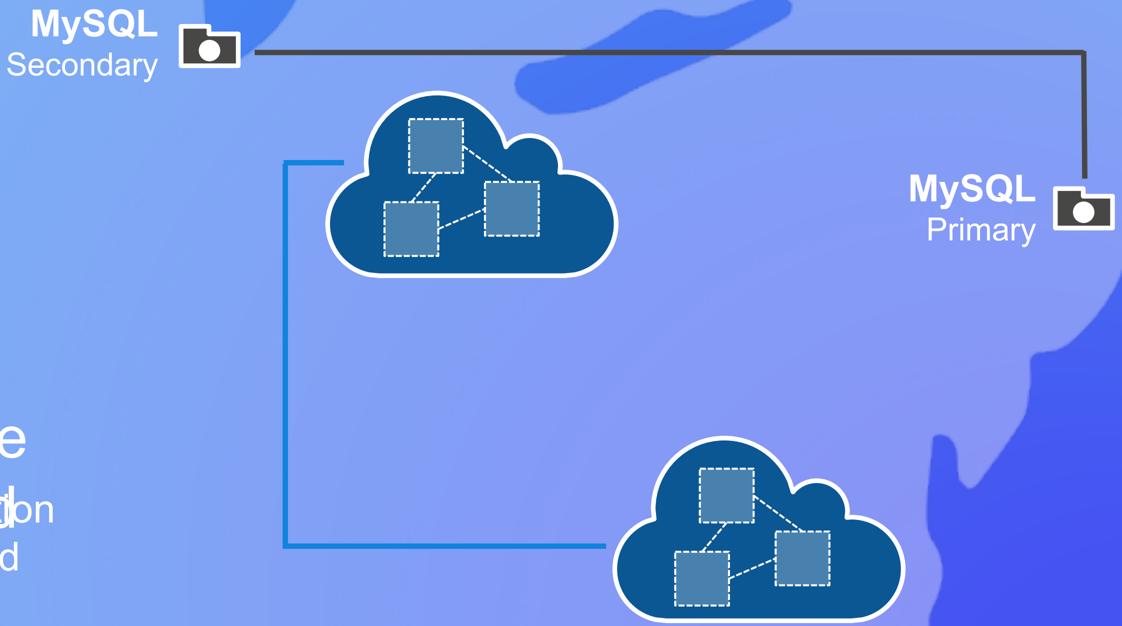


Regions and Zones

Datacenter Native

Migration to Cloud

Keep the same configuration and run MySQL on a cloud instance yourself.

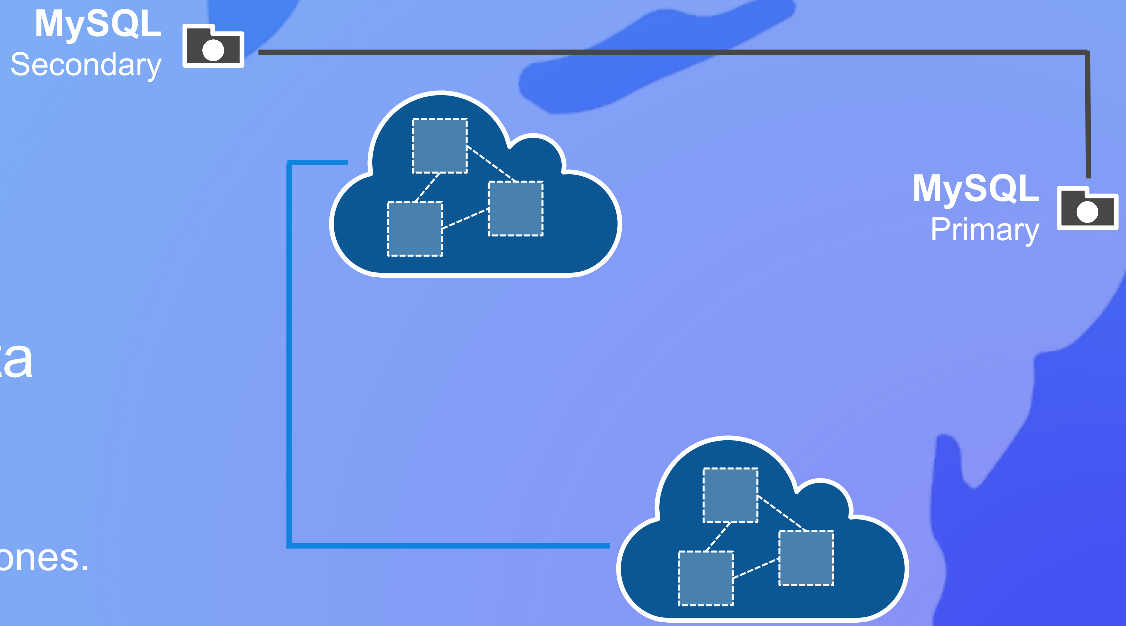


Regions and Zones

Cloud Native Data Migration

AWS Aurora

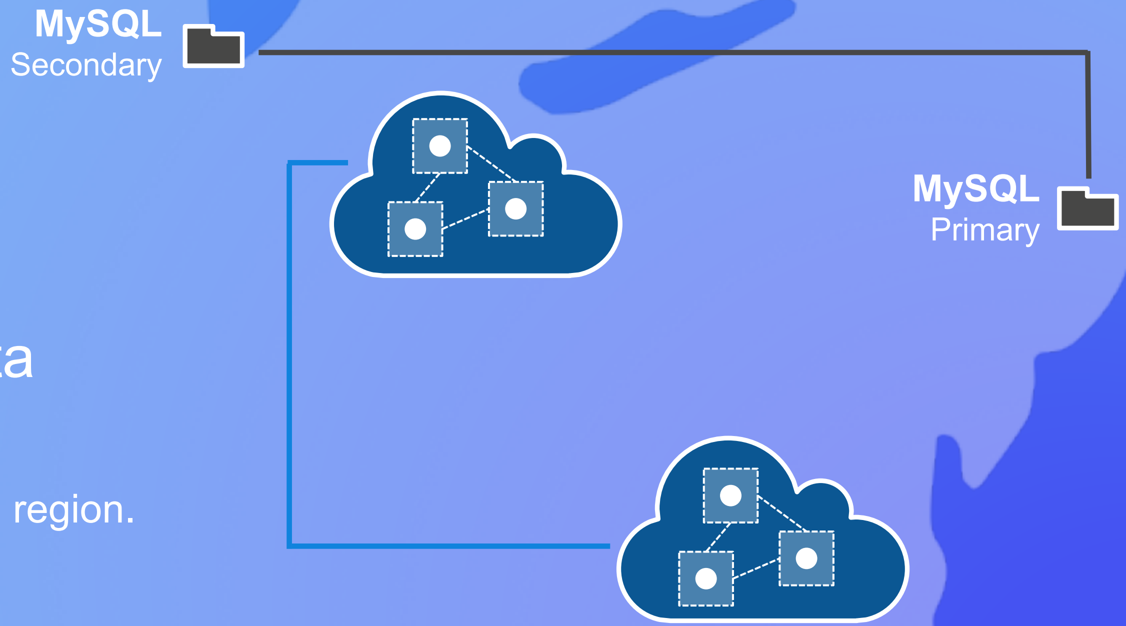
Distribute over all three zones.



Regions and Zones

Cloud Native Data Migration

More resilient within each region.



Cloud Native Principle

Distribute over zones within a region by default.

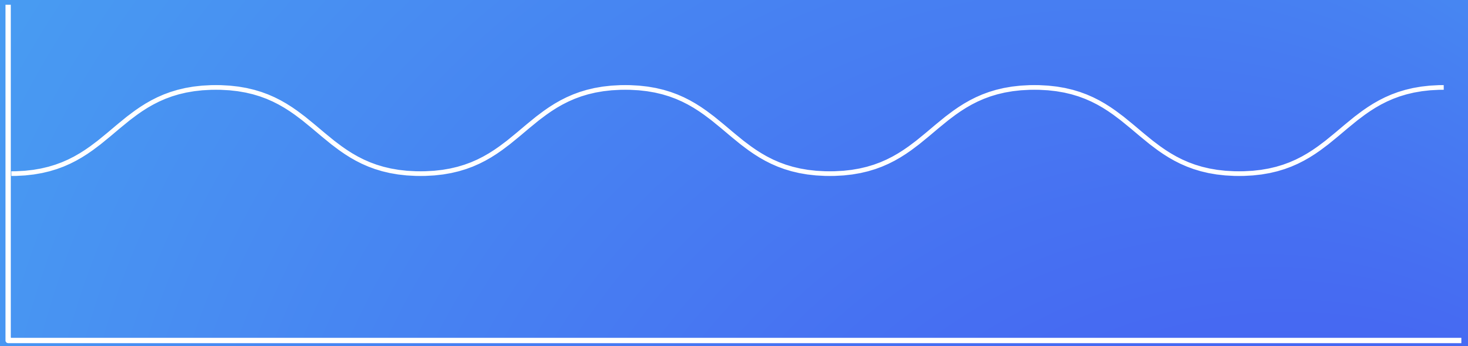
Elasticity

DATACENTER

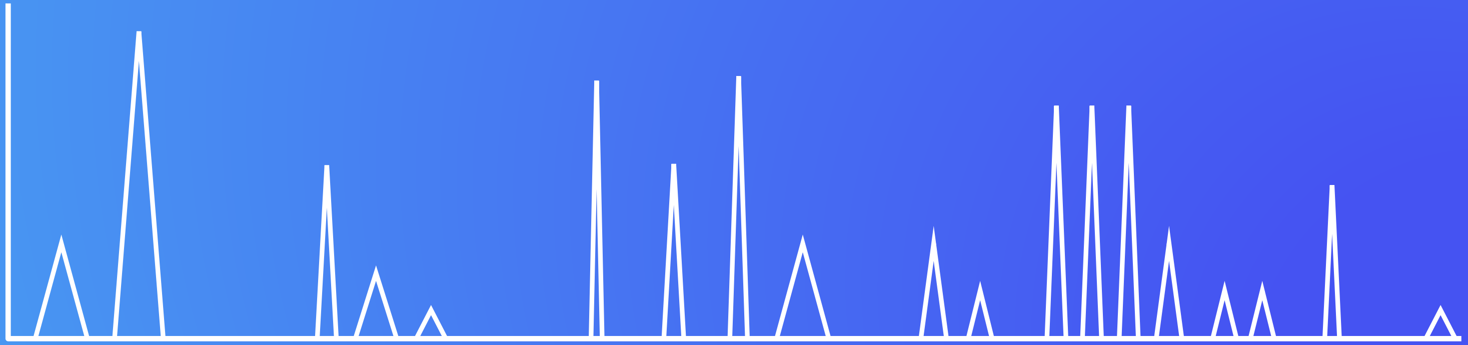
Hard to get over 10% utilization—
need extra capacity in case of peak.

CLOUD

Target over 40% utilization—
no capacity overload issues.



Autoscaling for predictable heavy workloads



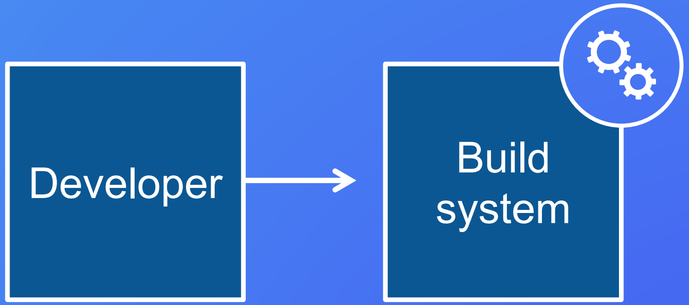
Serverless for spiky workloads with idle periods

Cloud Native Principle

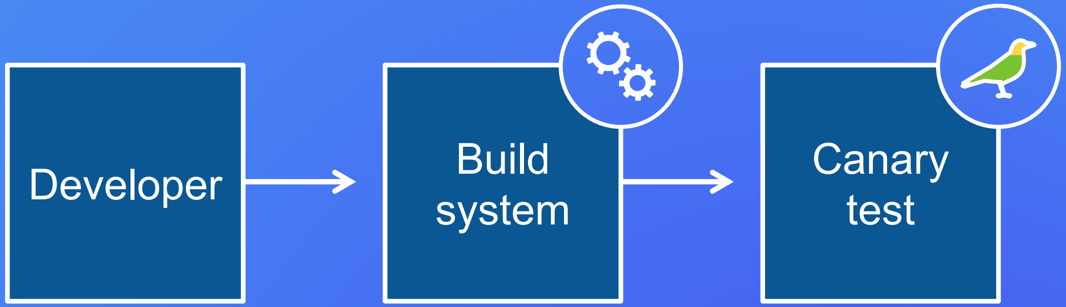
Turn it off when it’s idle.

Many times higher utilization
Huge cost savings
Avoids capacity overloads

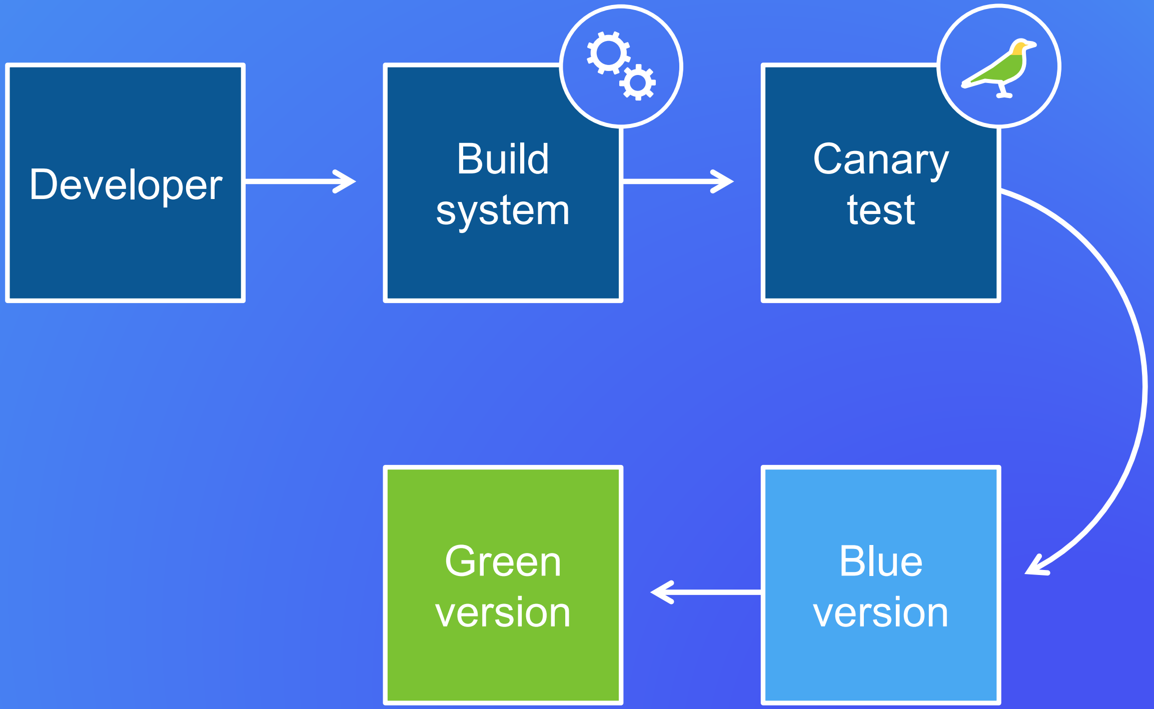
Versioned delivery pipeline



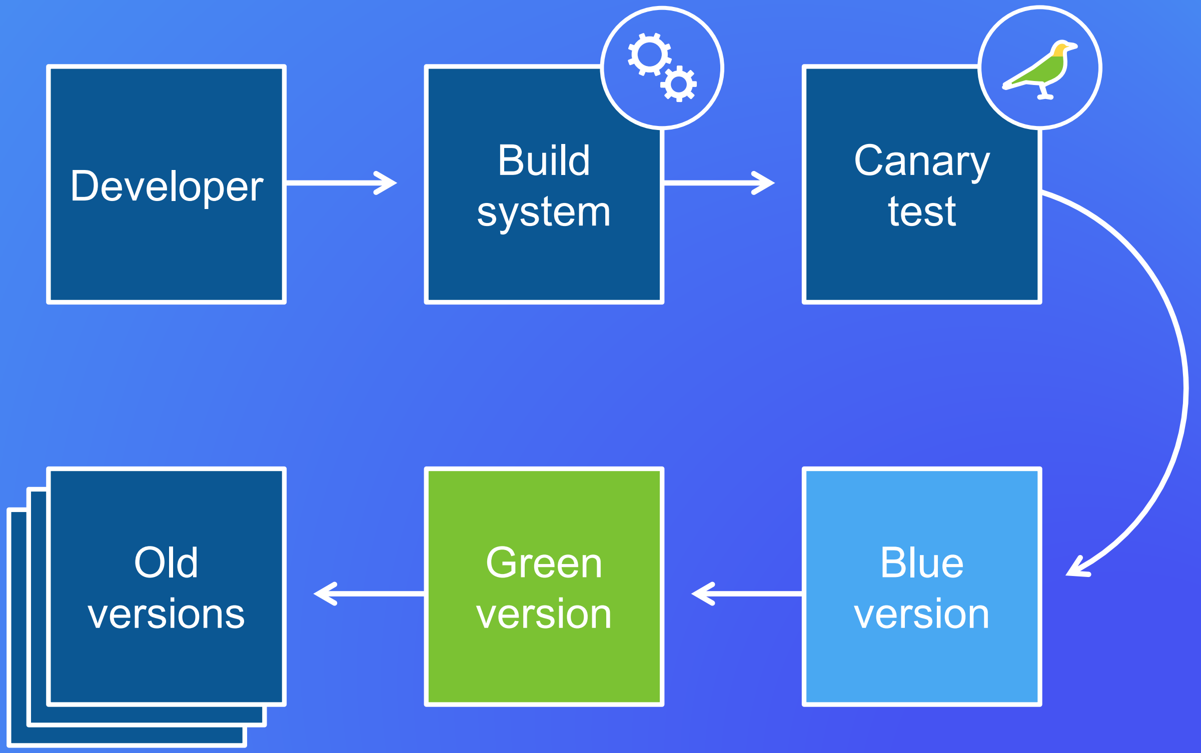
Versioned delivery pipeline



Versioned delivery pipeline



Versioned delivery pipeline



Cloud Native Principle

Immutable code.

Automated builds
Ephemeral instances, containers, and functions
Blue–Green deployments
Versioned services

Cloud Native Principle

Pay as you go, afterwards

Self service—no waiting

Globally distributed by default

Cross-zone/region availability models

High utilization—turn idle resources off

Immutable code deployments

Principles

Practice

Cloud Native Practice 2012

Netflix OSS

Instances

Java focus → Spring cloud today

Cloud Native Practice 2014

Docker

Containers

Golang focus → Kubernetes today

Cloud Native Practice 2016

AWS Lambda

Functions and events

Node.js focus → Serverless today

Pioneers

Serverless

Fastest development

Low cost

Tooling emerging

Settlers

Containers

Efficient

Faster

Too many choices

Rapidly evolving tooling

Town Planners

Instances

Risk adverse

Safe but slow

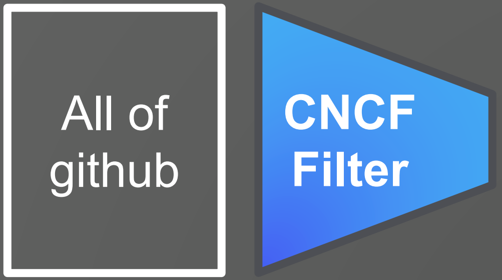
Mature tooling

Too many choices
Rapidly evolving tooling

CNCF

Cloud Native Computing Foundation

A curated collection of interesting open source projects that have broad support



Kubernetes
Orchestration

Prometheus
Monitoring

OpenTracing
Tracing

Fluentd
Logging

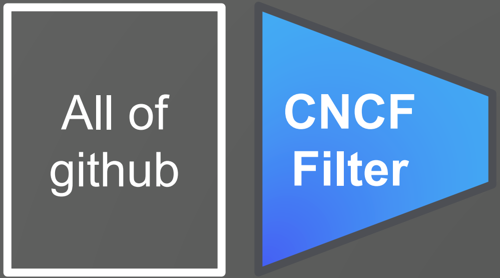
linkerd
Service Mesh

gRPC
Remote Procedure Call

CNCF

Cloud Native Computing Foundation

A curated collection of interesting open source projects that have broad support



Kubernetes Orchestration	**Prometheus** Monitoring	**CoreDNS** Service Discovery
OpenTracing Tracing	**Fluentd** Logging	**Containerd** Container Runtime
linkerd Service Mesh	**gRPC** Remote Procedure Call	**rkt** Container Runtime
CNI Networking	**Envoy** Service Mesh	**Jeager** Distributed Tracing

AWS (and everyone else) joined CNCF

Promote Cloud Native to enterprise customers

Integrate CNCF components into AWS ECS – CNI, containered, etc.

Integrate Kubernetes with AWS – installers, IAM, security, etc.

CNCF serverless working group

Kubernetes	VS	AWS ECS
Managed by customers		Managed for you by AWS
Single tenant install		Multi tenant service
Control plane overhead		Just EC2 instances by the second
Version upgrade management		Doesn’t apply
Networking: CNI		Moving to CNI
IAM integration fixes needed		IAM Integrated

Kubernetes

Better developer features and APIs today

Improving operational features

Improving AWS integration

ECS

Better operational features today

Improving developer APIs – converging with CNCF components

Improving portability for applications

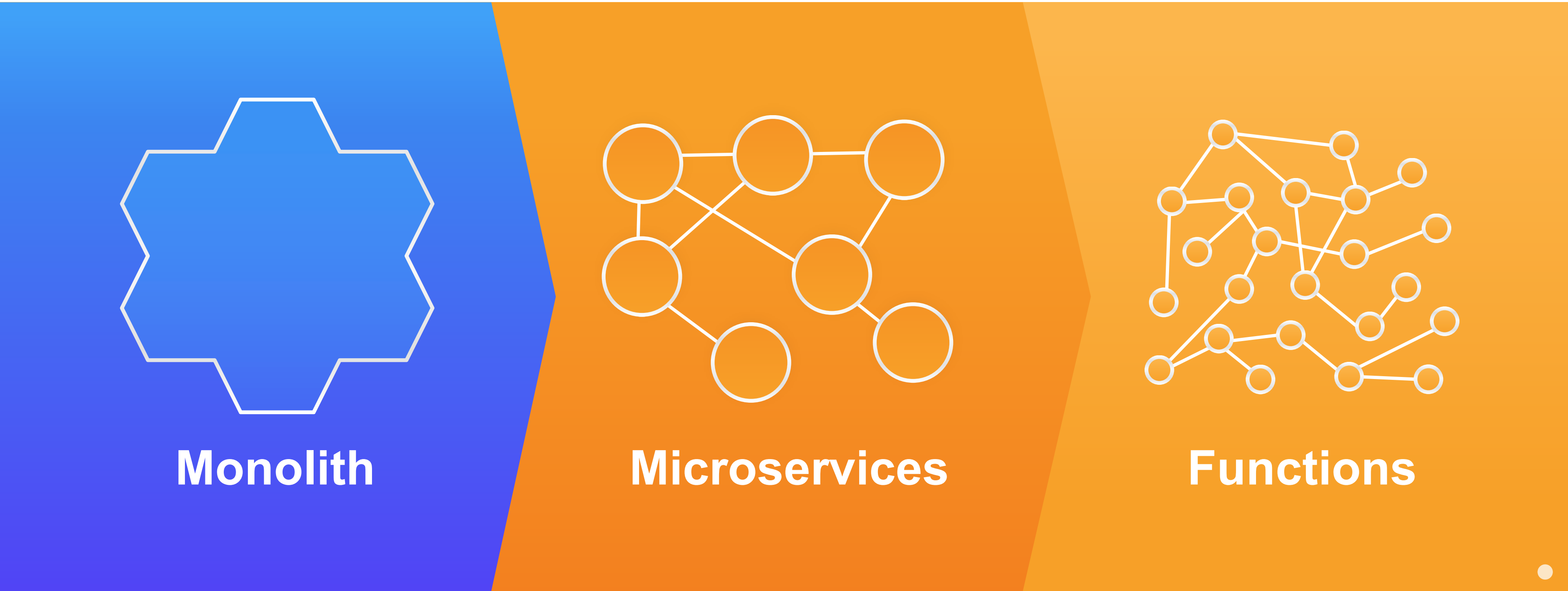
Serverless

Finish building and deploying the application in less time than you spent evaluating container runtimes…

Cloud Native Principles

Remain constant as practices evolve.

Evolution of Business Logic



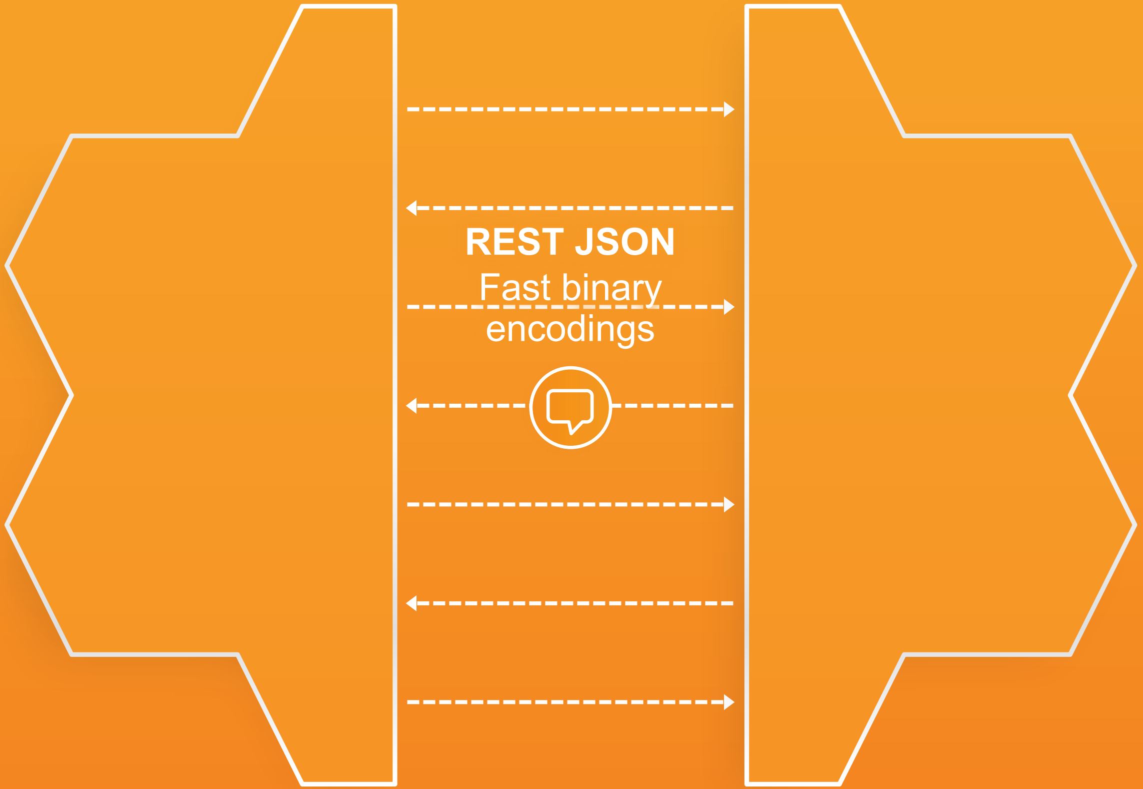
Splitting Monoliths Ten Years Ago

Splitting Monoliths Ten Years Ago

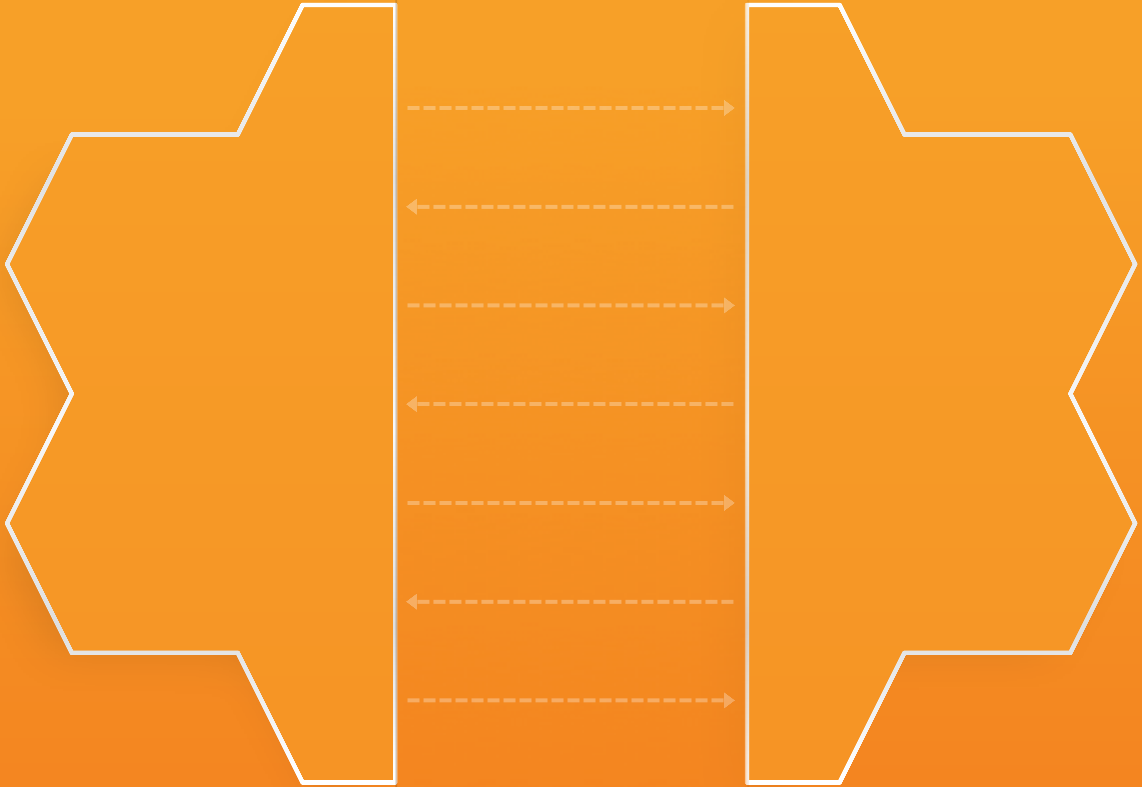
XML & SOAP
?xml ve
<categor
<conte

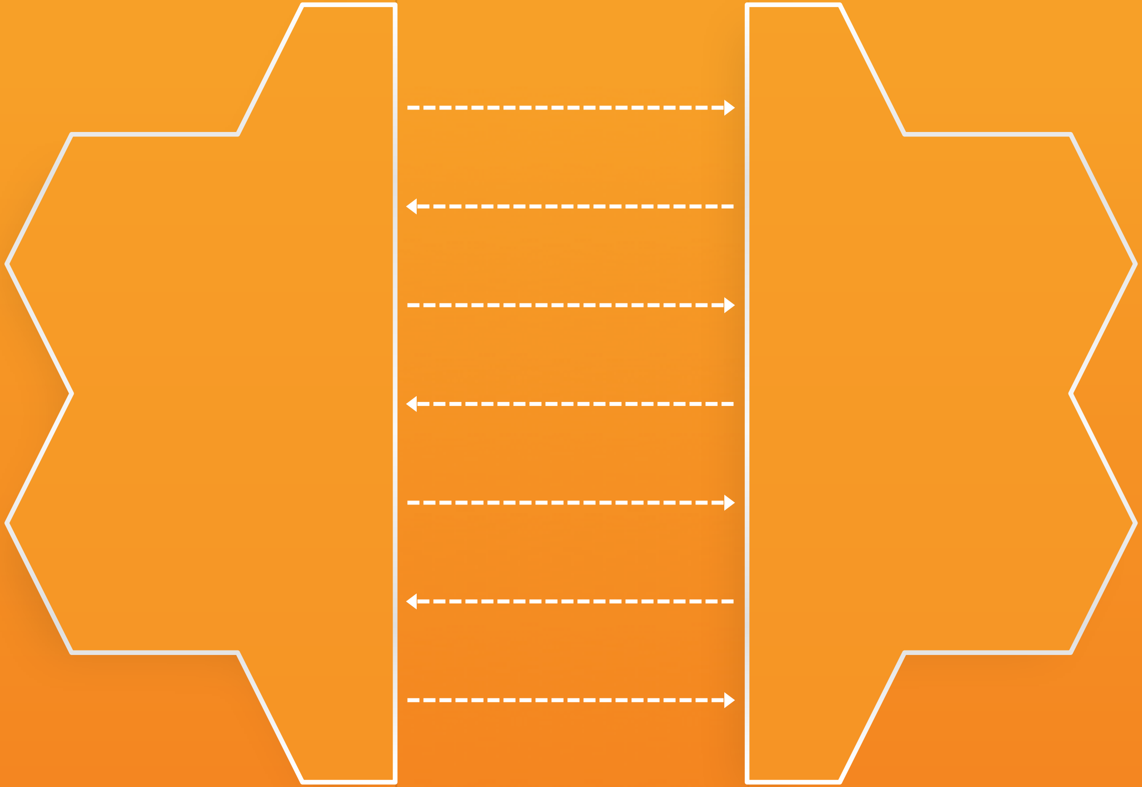
Splitting Monoliths Five Years Ago

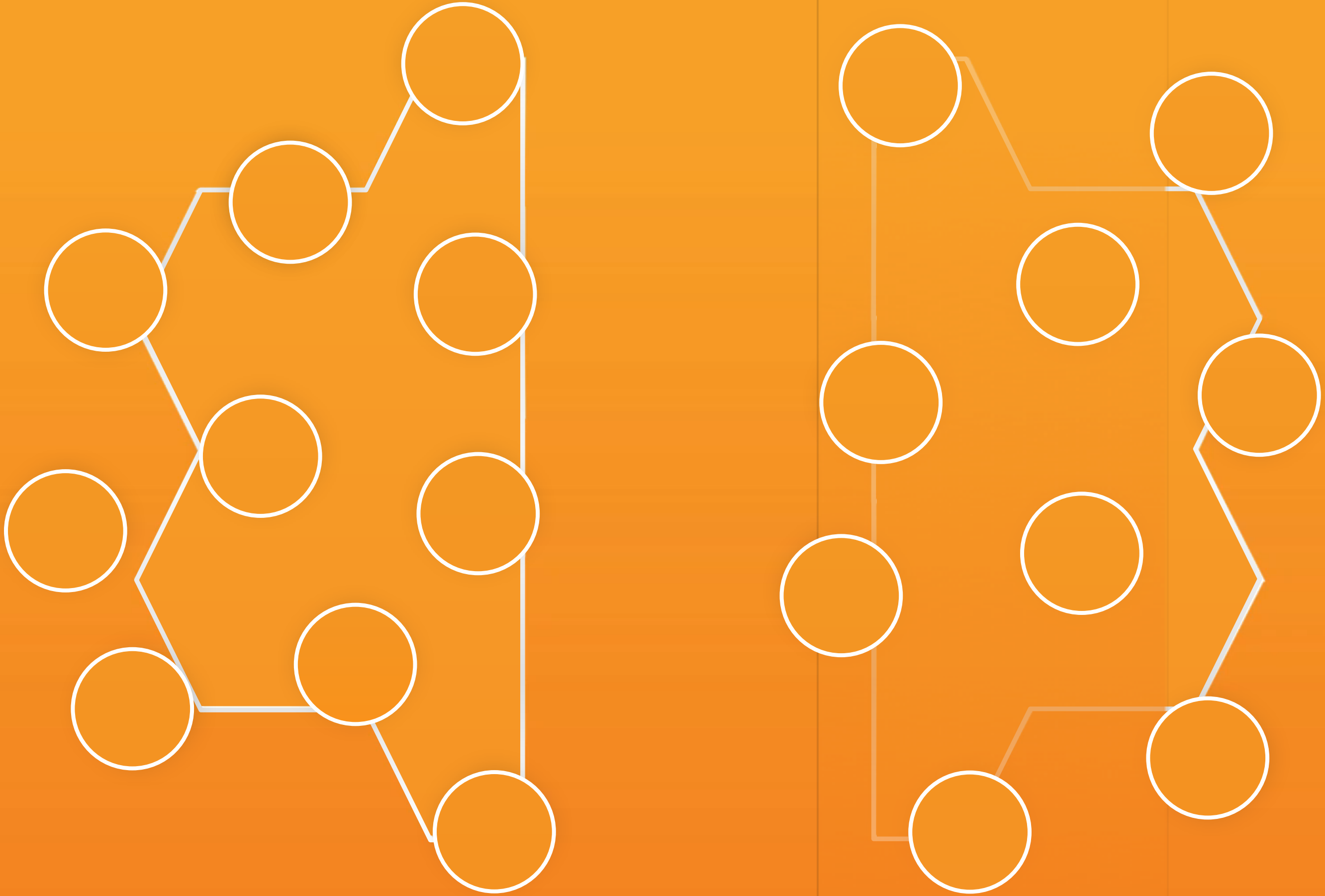
Splitting Monoliths Five Years Ago

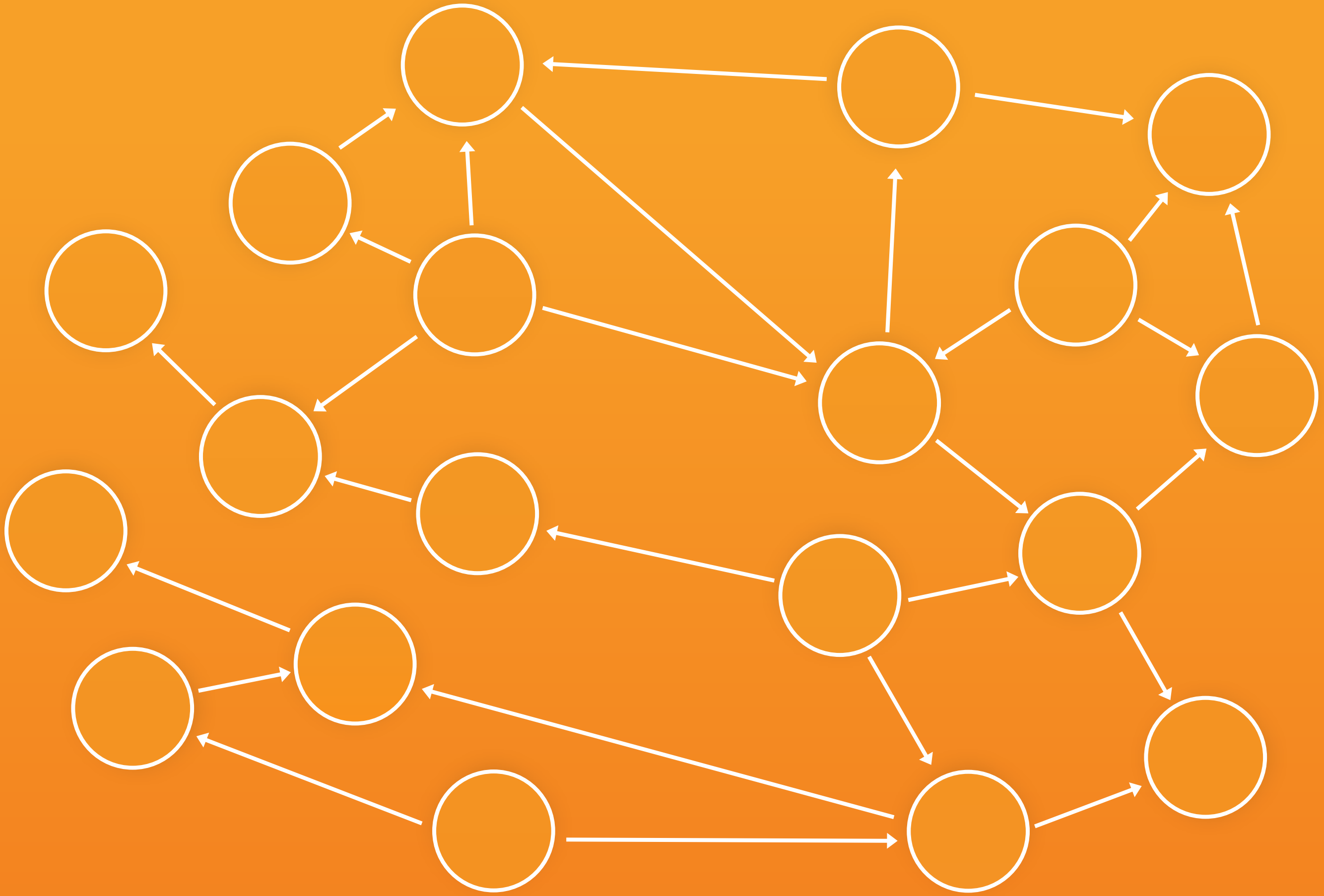


Splitting Monoliths Five Years Ago

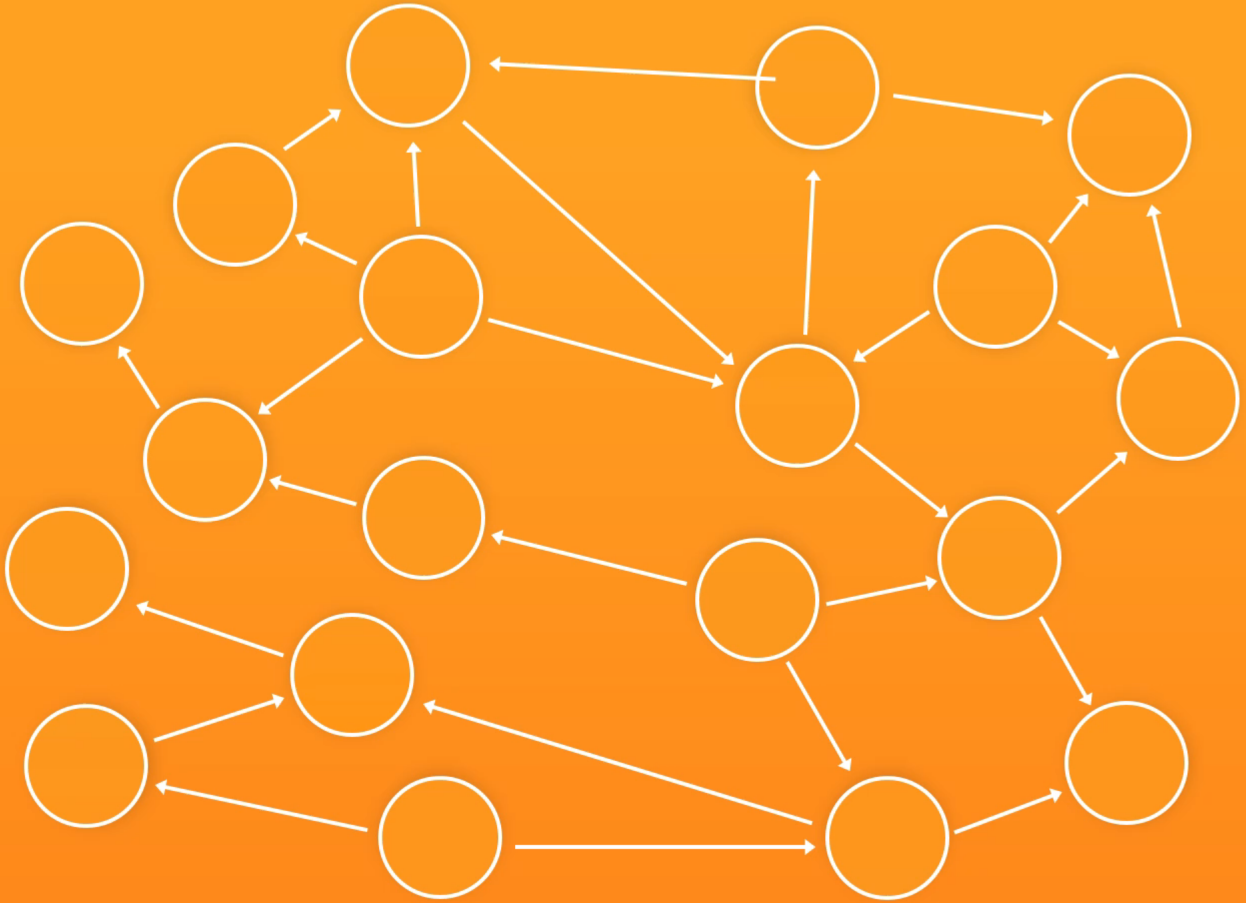








Microservices Five Years Ago



Microservices
Five Years Ago
to Functions

Standard building brick services provide standardized platform capabilities

Amazon SQS

Amazon API Gateway

Amazon DynamoDB

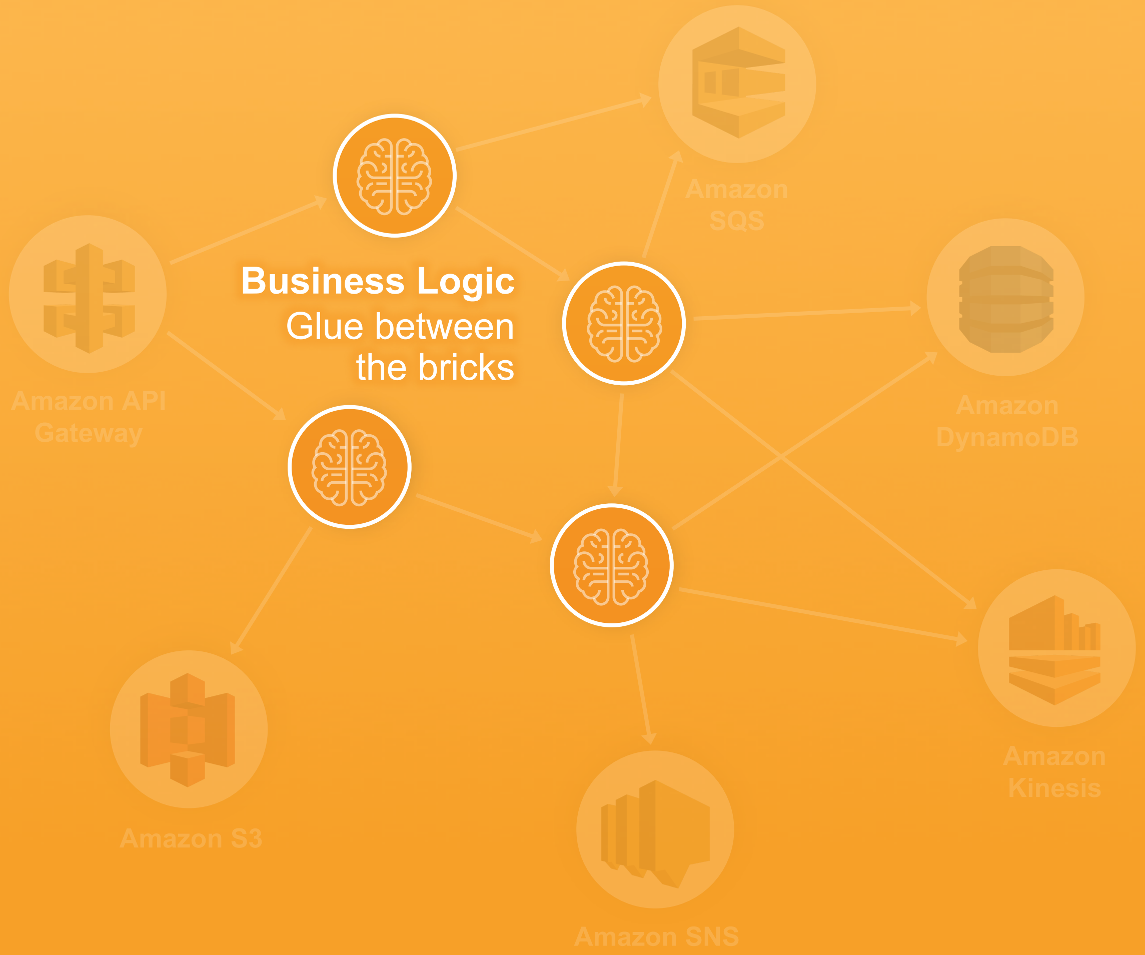
Amazon Kinesis

Amazon S3

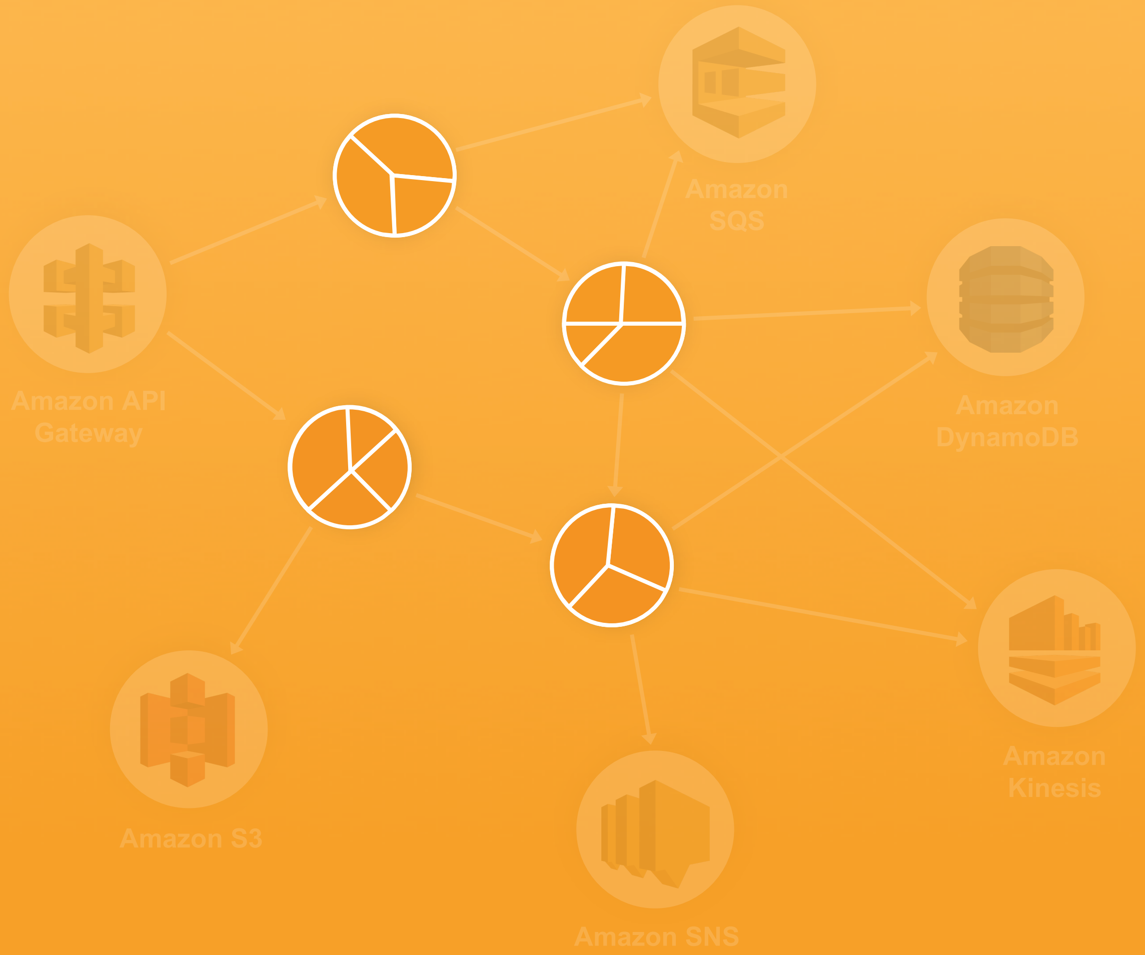
Amazon SNS

Microservices to Functions

Standard building brick services provide standardized platform capabilities



Microservices to Functions

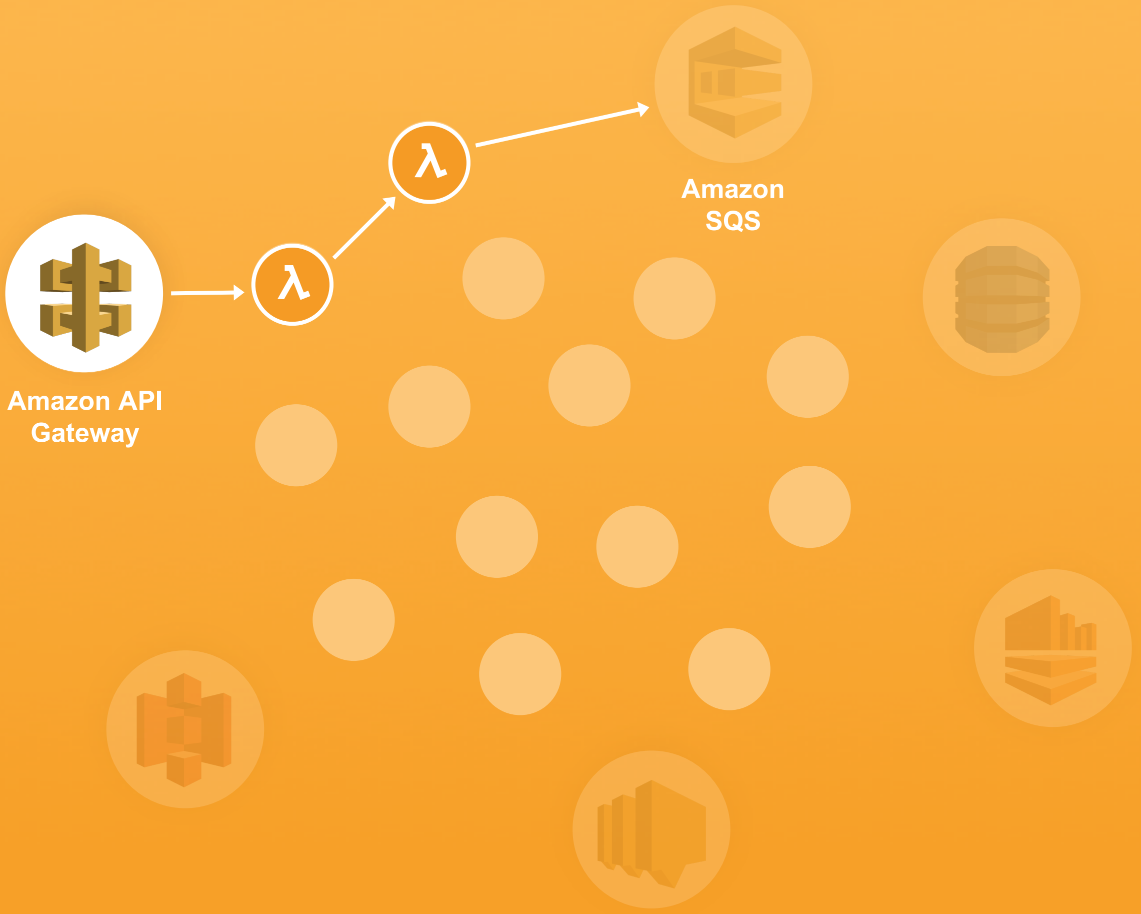


Microservices to Functions

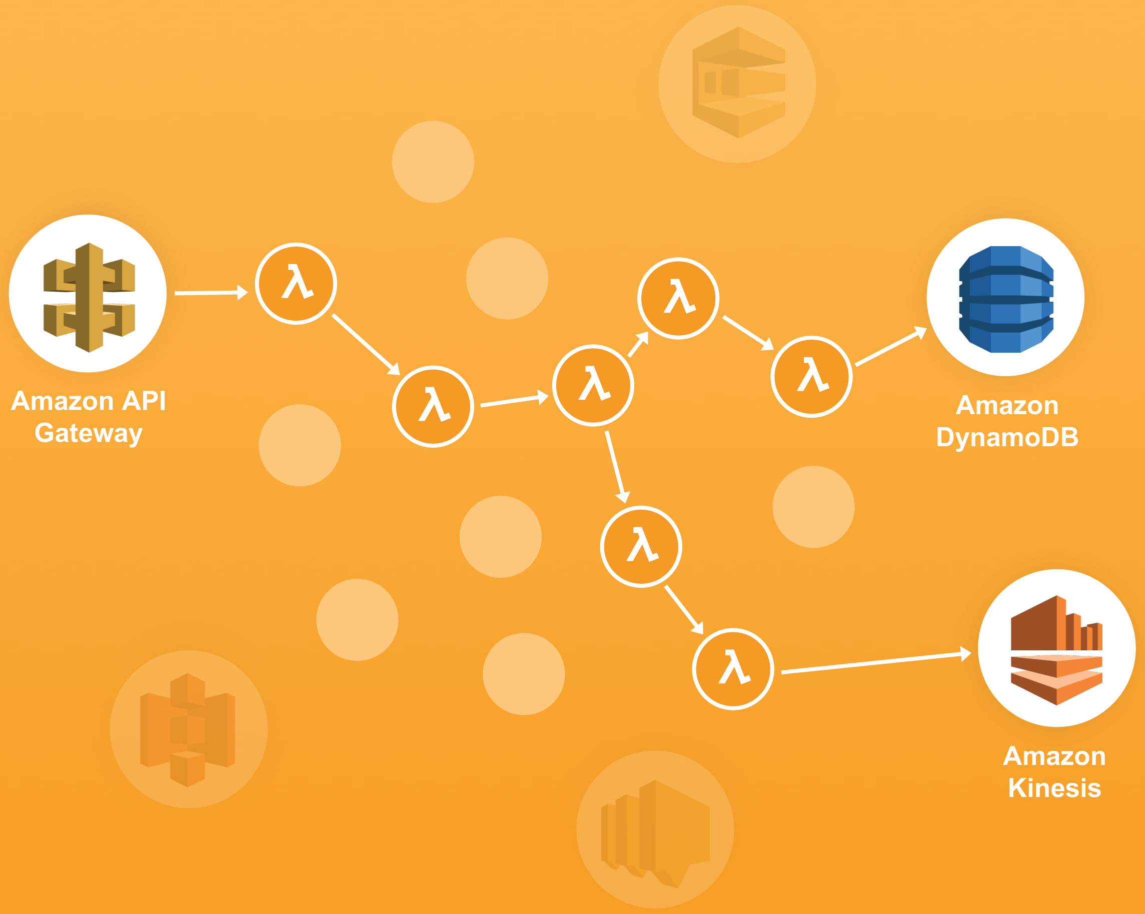
Microservices to Functions Ephemeral

Microservices to Ephemeral Functions

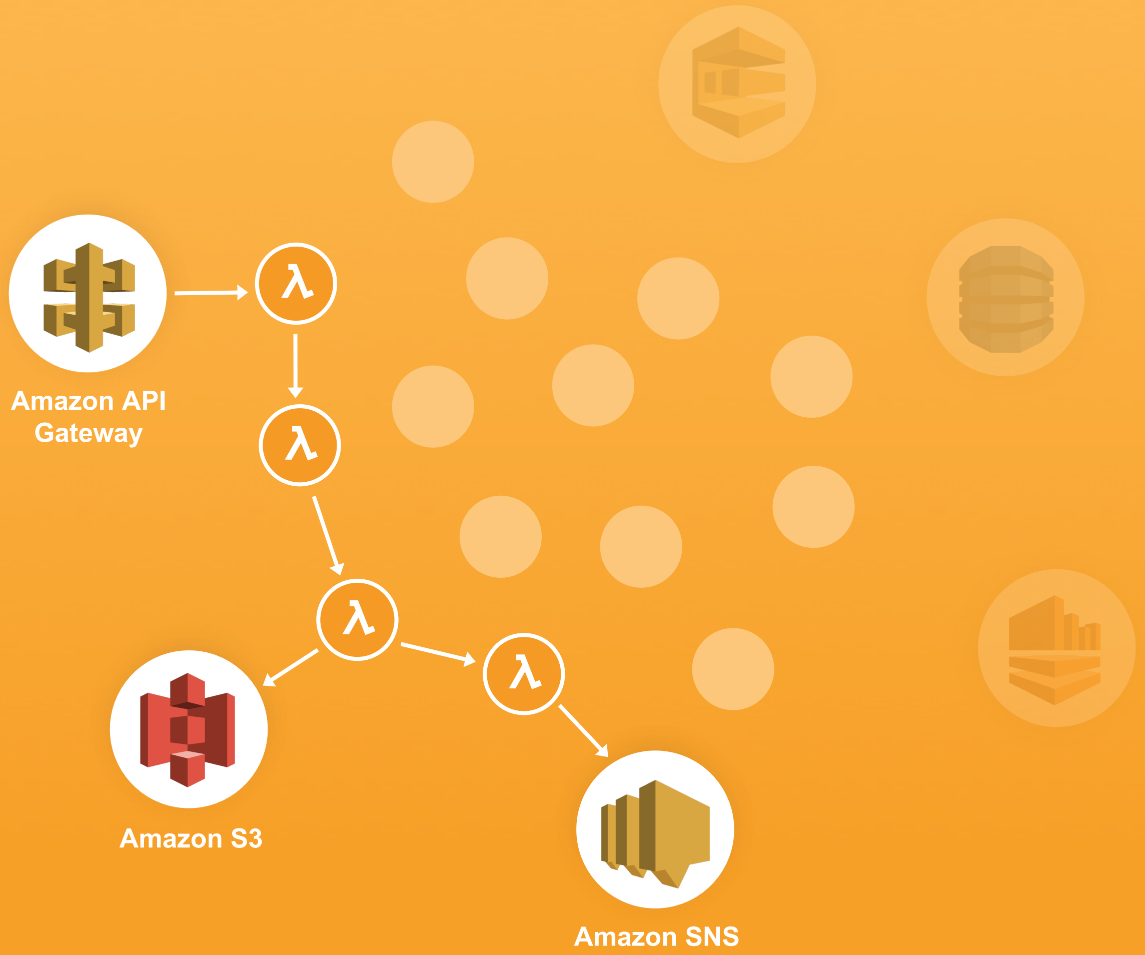
Microservices to Ephemeral Functions



Microservices to Ephemeral Functions



Microservices to Ephemeral Functions

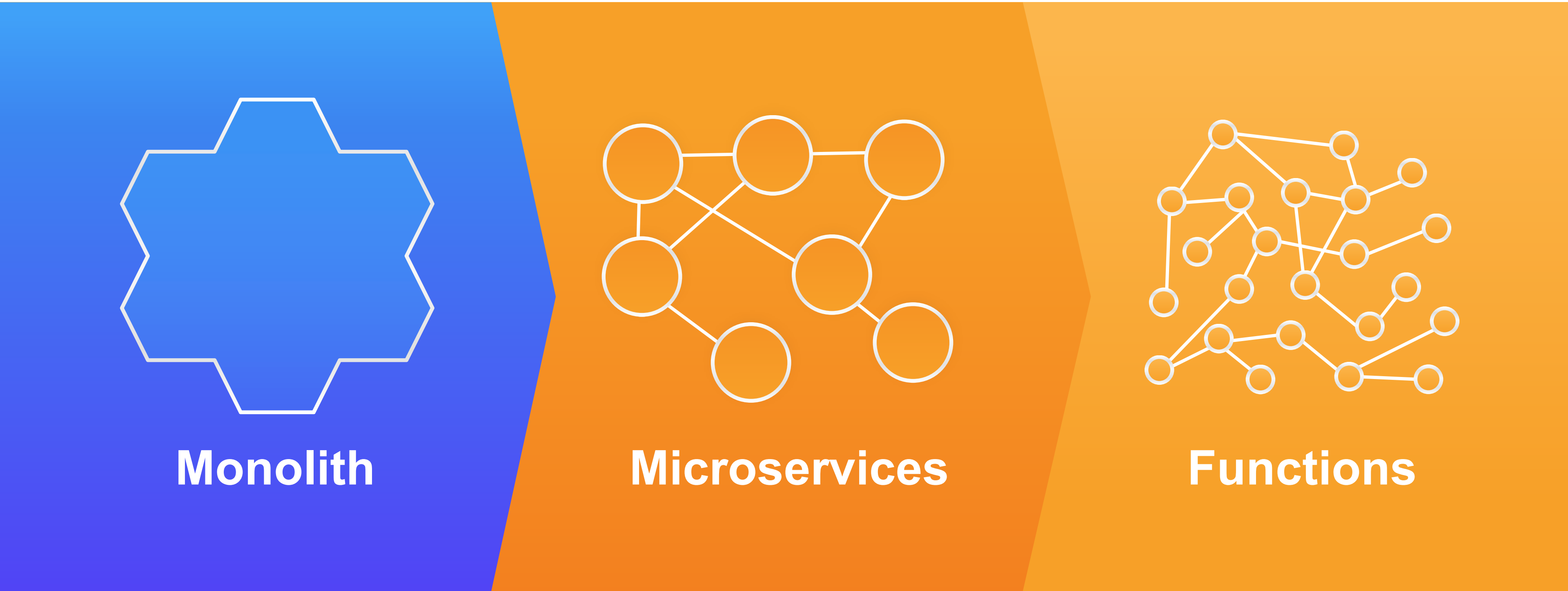


Microservices to Ephemeral Functions

Amazon SQS
Amazon API Gateway
Amazon DynamoDB
Amazon S3
Amazon Kinesis
Amazon SNS

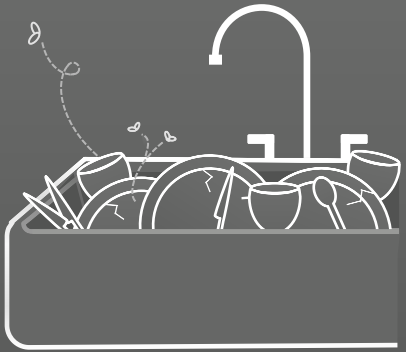
When the system is idle, it shuts down and costs nothing to run

Evolution of Business Logic



The New De-Normal

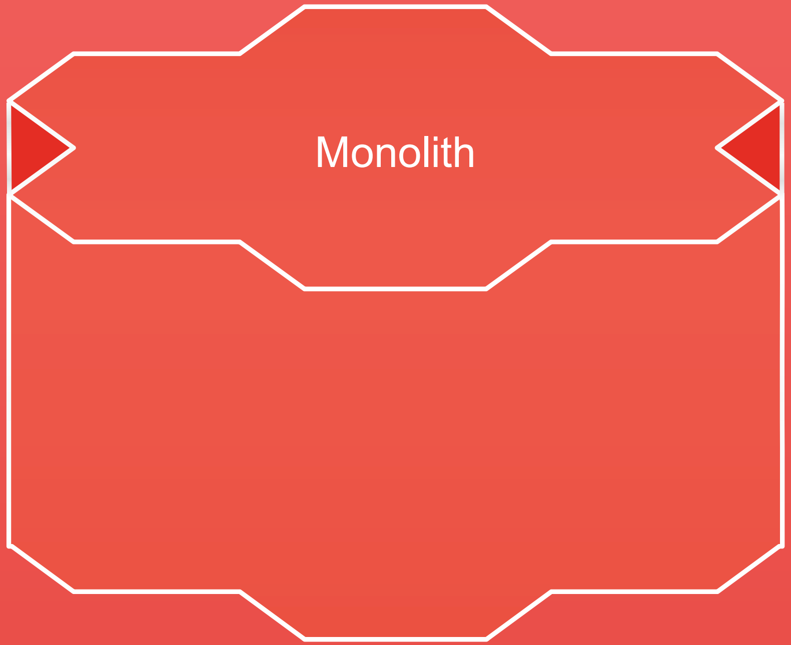
Monolithic Databases



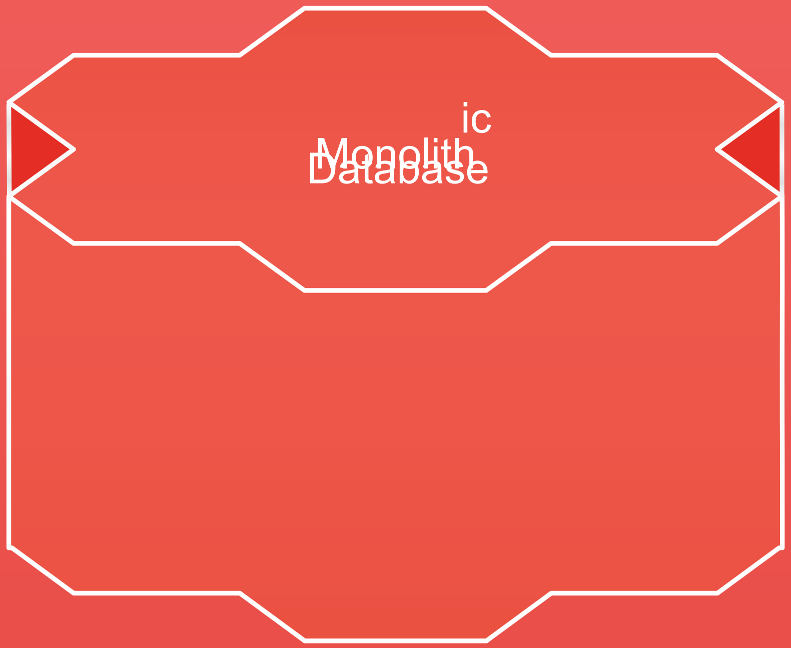
Kitchen Sink Analogy

De-normalized

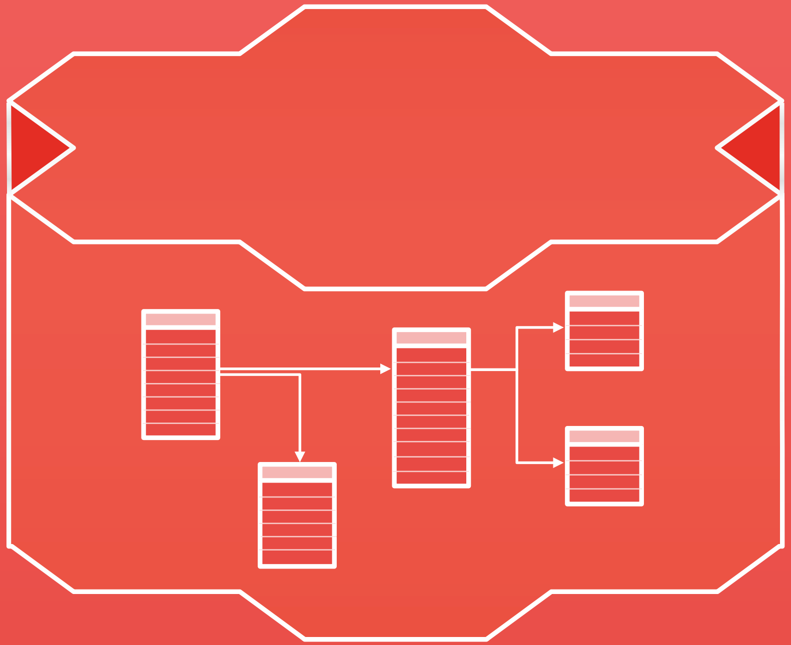
Expensive, Hard to Create and Run



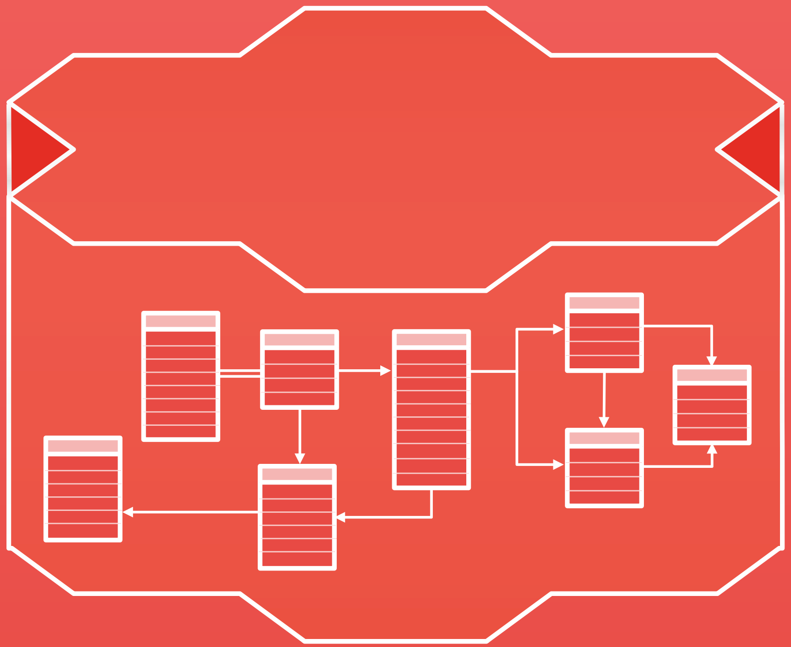
Expensive, Hard to Create and Run



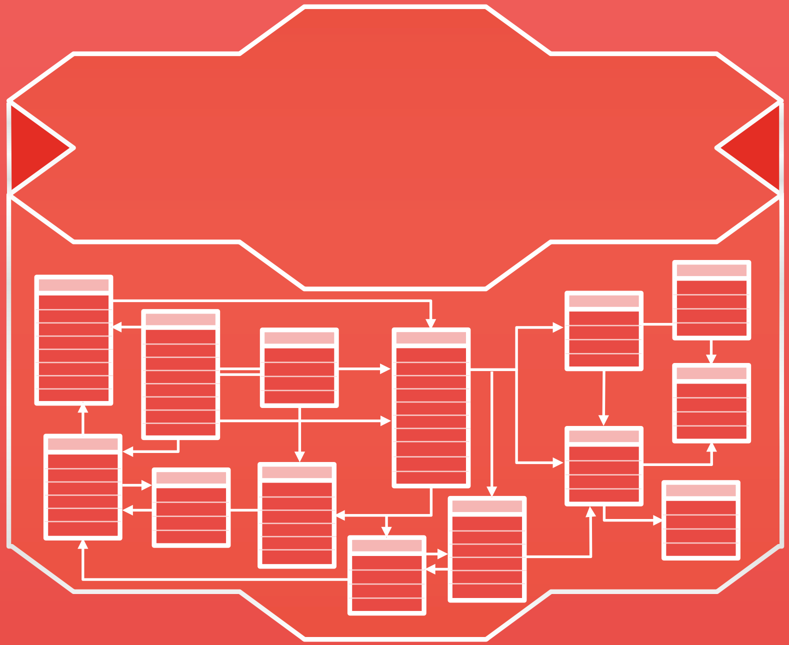
Database Schema
Entity Relationship



Database Schema
Entity Relationship



Database Schema
Entity Relationship

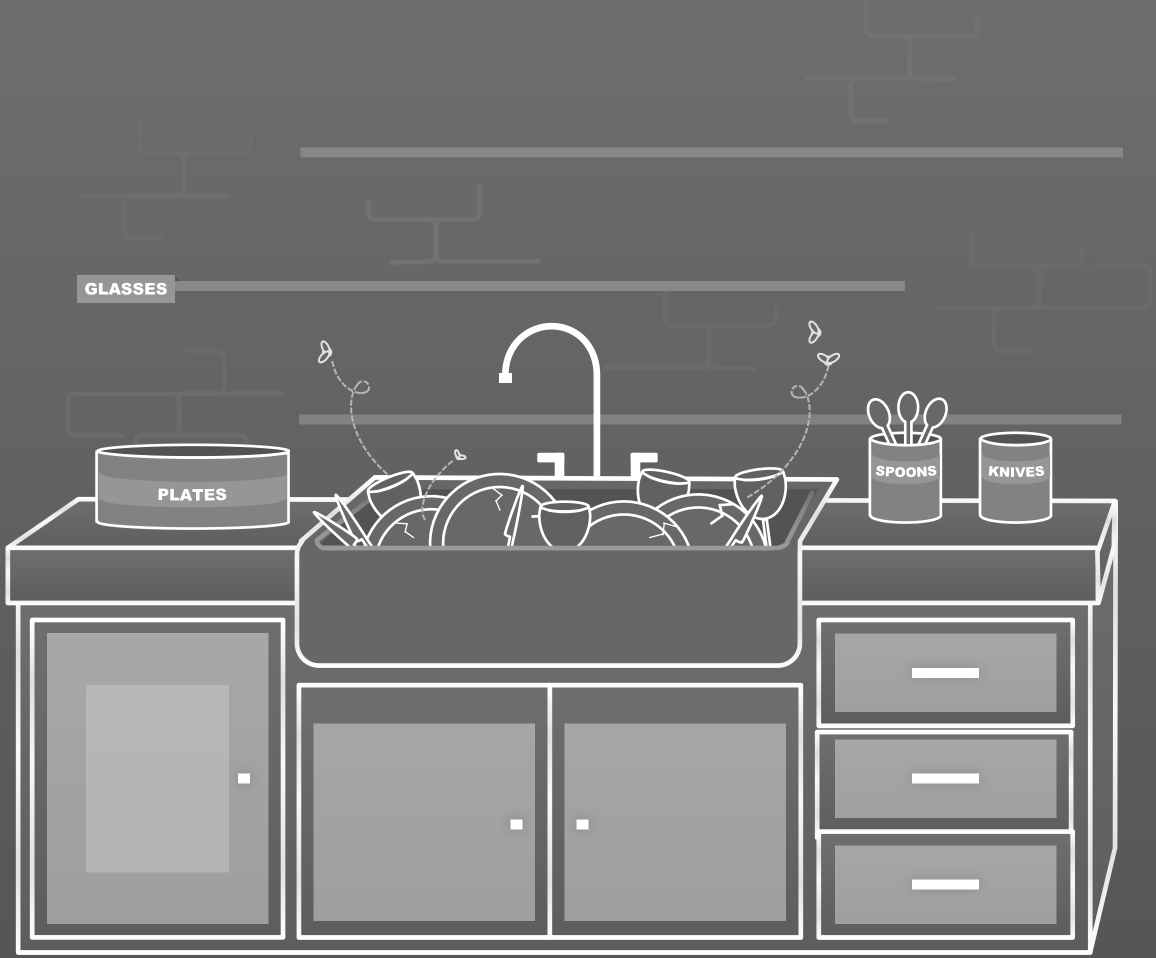


Kitchen Sink Analogy

Kitchen Sink Cleanup

Kitchen Sink Cleanup

Kitchen Sink Cleanup



Kitchen Sink Cleanup

Kitchen Sink Cleanup

Kitchen Sink Cleanup

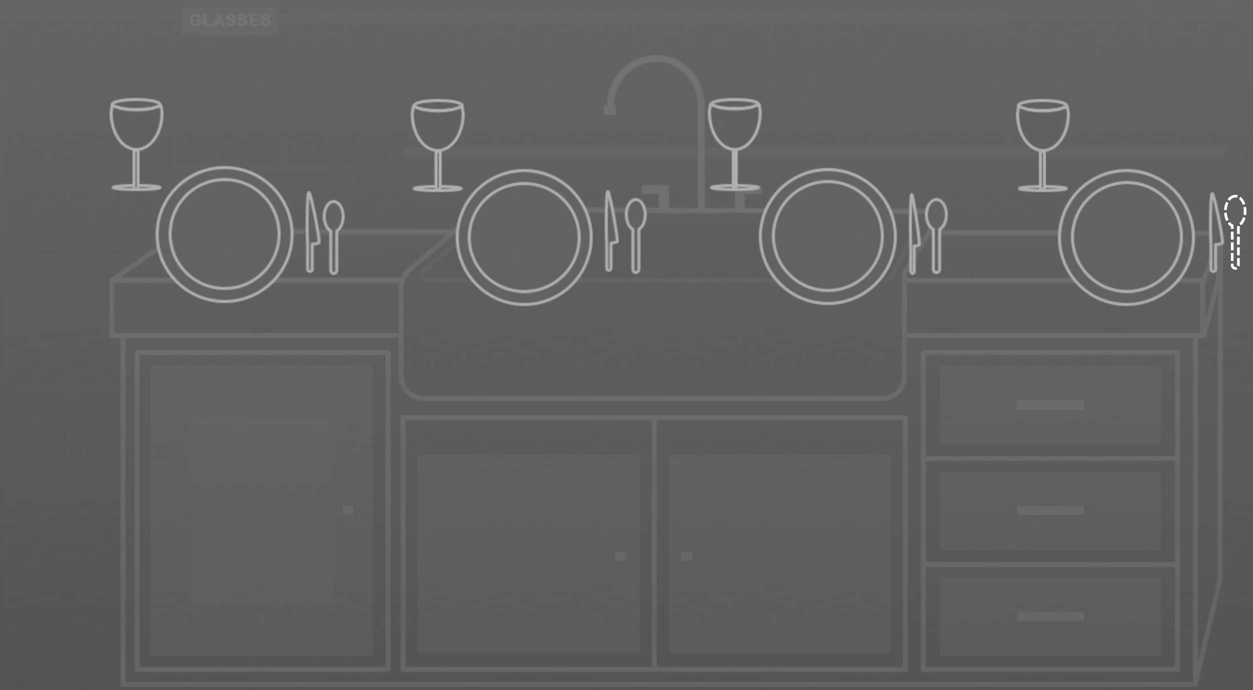
Kitchen Sink Cleanup

Consistency Problem

How Many Complete Sets Are There?

Consistency Problem

How Many Complete Sets Are There?

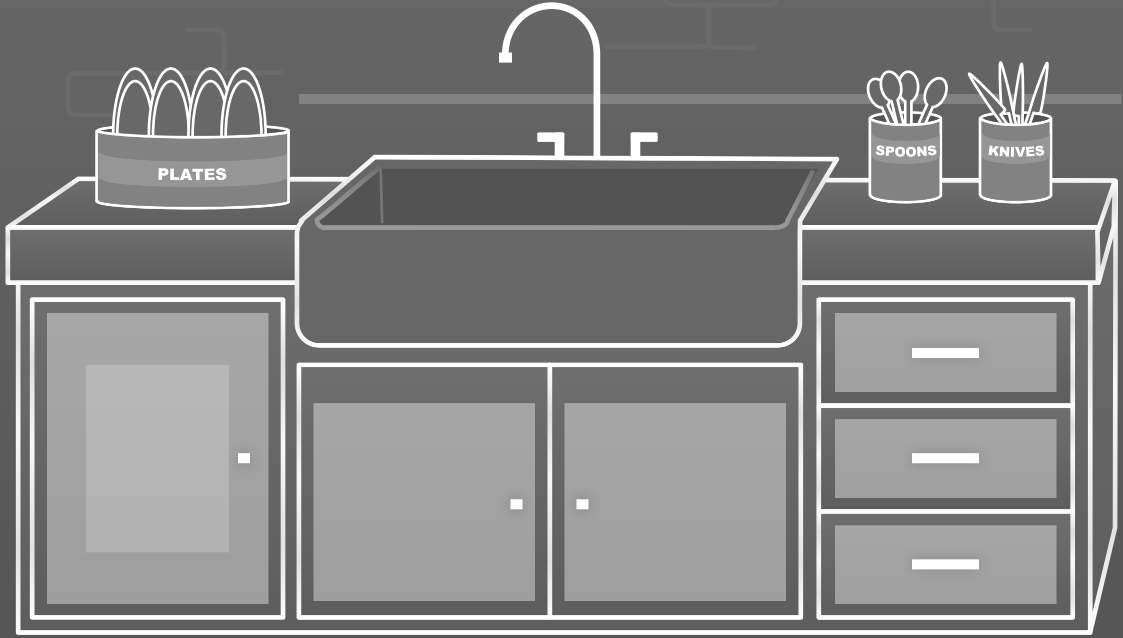


Consistency Problem

How Many Complete Sets Are There?

GLASSES
PLATES
SPOONS
KNIVES

Adding a New Use Case



Adding a New Use Case

Cloud Makes it Easy to Add New Databases

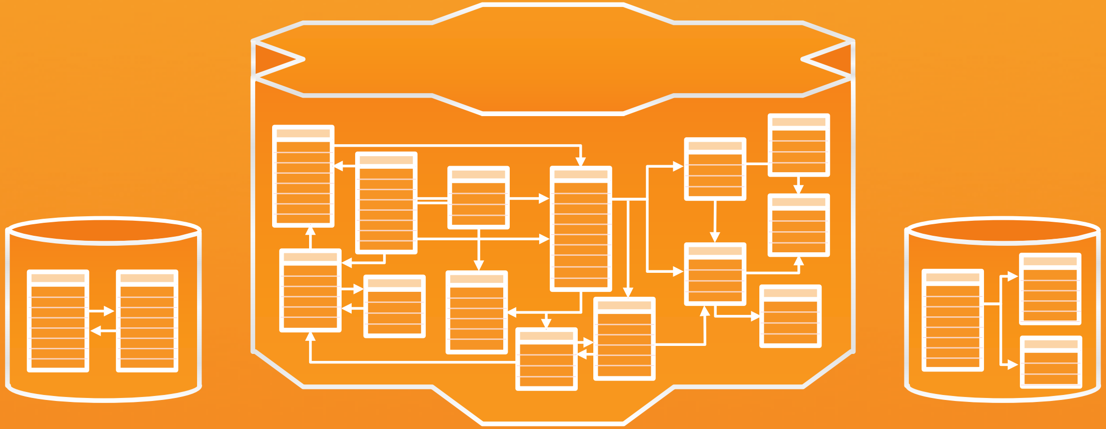
Amazon DynamoDB

Amazon DMS

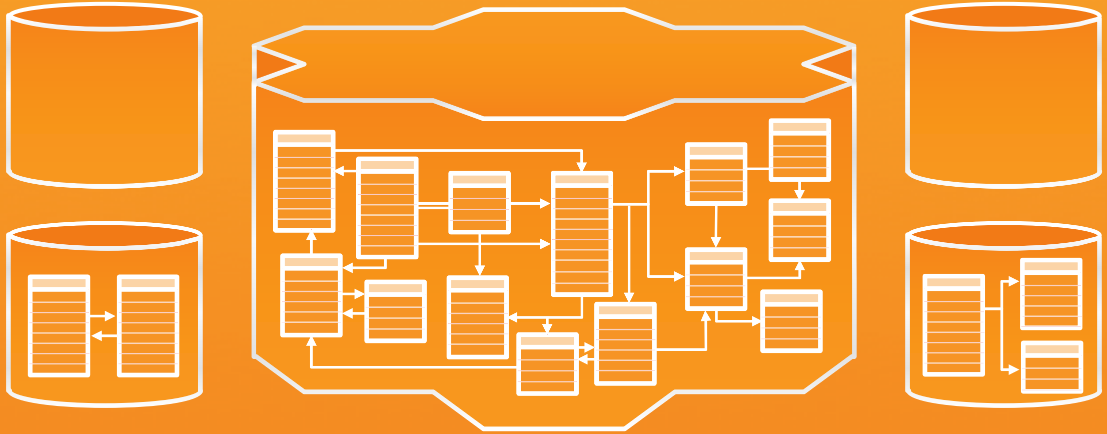
Amazon RDS

Amazon Redshift

Amazon Aurora for MySQL and Postgres



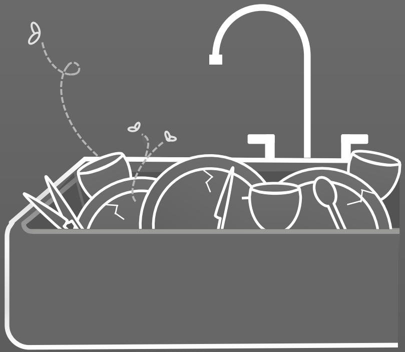
Untangle and Migrate Existing “Kitchen Sink” Schemas



Untangle and Migrate Existing "Kitchen Sink" Schemas

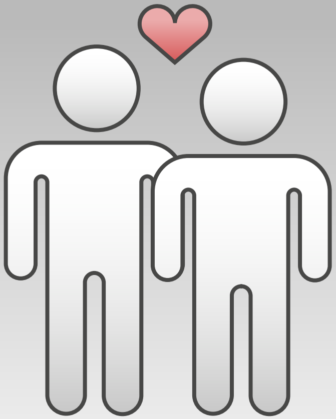
The New De-Normal

Monolithic Databases


Kitchen Sink Analogy

De-normalized

Lock-in and the Lifecycle of Dependencies



Choosing, Using and Losing

What is the return on investment (ROI) for each phase?

Choosing

Using

Losing

What is the ROI for each phase?

How has ROI changed with advances in technology and practices?

Choosing

Using

Losing

Choosing

Using

Losing

Choosing

Investments

Negotiating, learning, experimenting

Hiring experts, building

Installing, customizing

Developing, training

Choosing

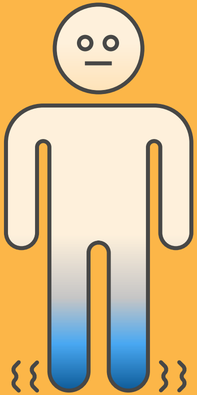
How much time elapses?

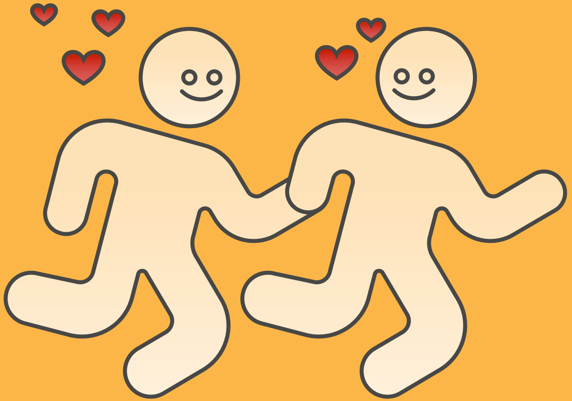
“The best decision is the right decision. The next best decision is the wrong decision. The worst decision is no decision.”

—Scott McNealy

Choosing

Analysis Paralysis vs. Snap Judgement





Choosing

Making a commitment

Whenever development is frozen, and the operations team takes over, the key is turned in the lock

Choosing—What Changed?

Old World	New World
Monolith—all in one	Microservice—fine grain
Proof of concept install	Web service/Open source
Enterprise purchase cycle	Free tier/free trial
Months	Minutes
$100K–$Millions	$0–$1000s

Choosing

Using

Losing

Using

Investments

Cost of setup

Cost of operation

Capacity planning

Scenario planning

Incident management

Tuning performance and utilization

Using

Returns

Service capabilities

Availability, functionality

Scalability, agility

Efficiency

Using - What Changed?

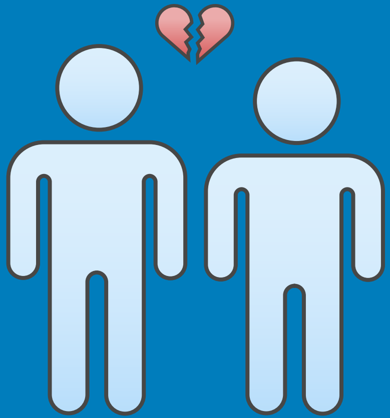
Old World	New World
Frozen installation	Continuous delivery
Ops specialist silo	Dev automation
Capacity upgrade costs	Elastic cloud resources
Low utilization	High utilization
High cost of change	Low cost of change

Choosing

Using

Losing

Losing



Investments

Negotiating time

Contract penalties

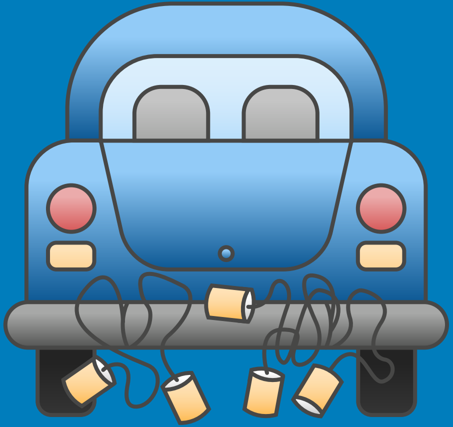
Replacement costs

Decommissioning effort

Archiving, sustaining legacy

Losing

Returns



Reduced spending

More advanced technology

Better service, agility, scalability

Choose again, the cycle continues…

Losing—What Changed?

Old World	New World
Monolithic—all or nothing	Microservices—fine grain
Frozen waterfall projects	Agile continuous delivery
Long term contracts	Pay as you go
Local dependencies	Remote web services

Old World	New World
Monolithic on-prem waterfall lock-in	Agile cloud-native micro-dependencies
Years	Weeks
Millions of dollars	Hundreds of dollars
100s of dev years	A few dev weeks
Lock-in	Refactoring
Lawyers and contracts	Self service

aws

Bottom line

ROI for choosing, using, losing has changed radically. Stop talking about lock-in, it’s just refactoring dependencies

The cost of each dependency is far lower

Frequency of refactoring is far higher

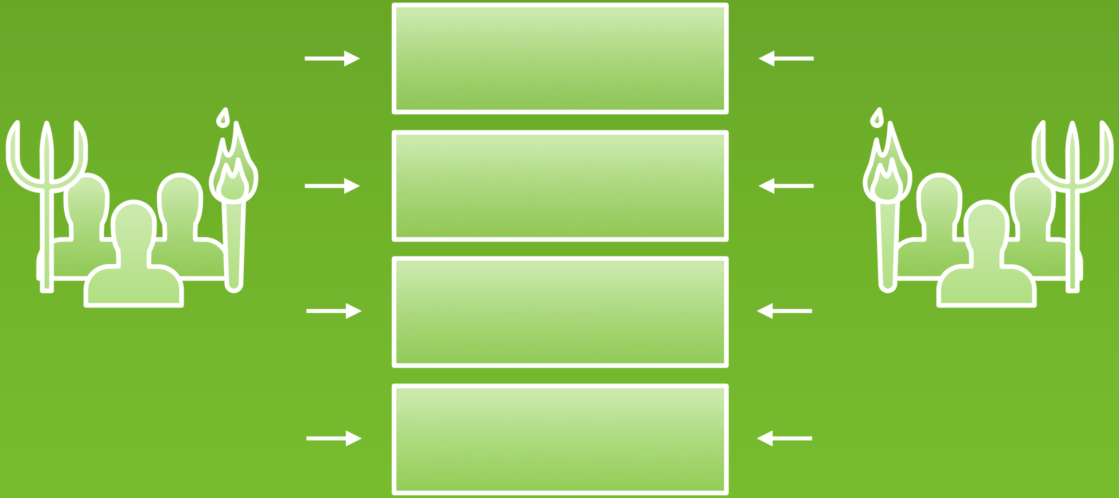
Investment and return is much more incremental

Chaos Architecture

A Cloud Native Availability Model

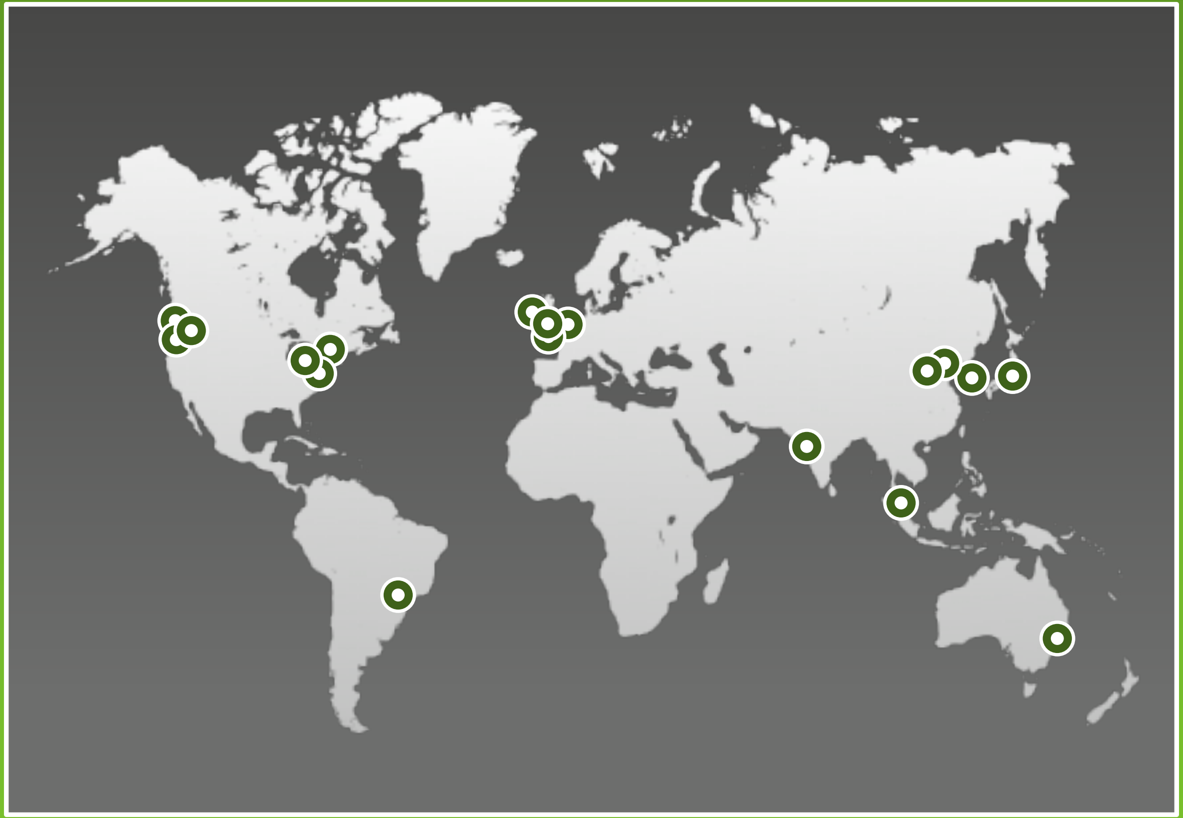
Chaos Architecture

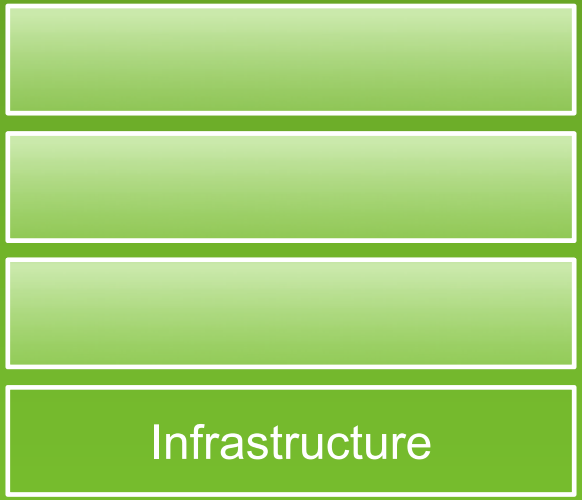
Four layers
Two teams
An attitude



Infrastructure and Services

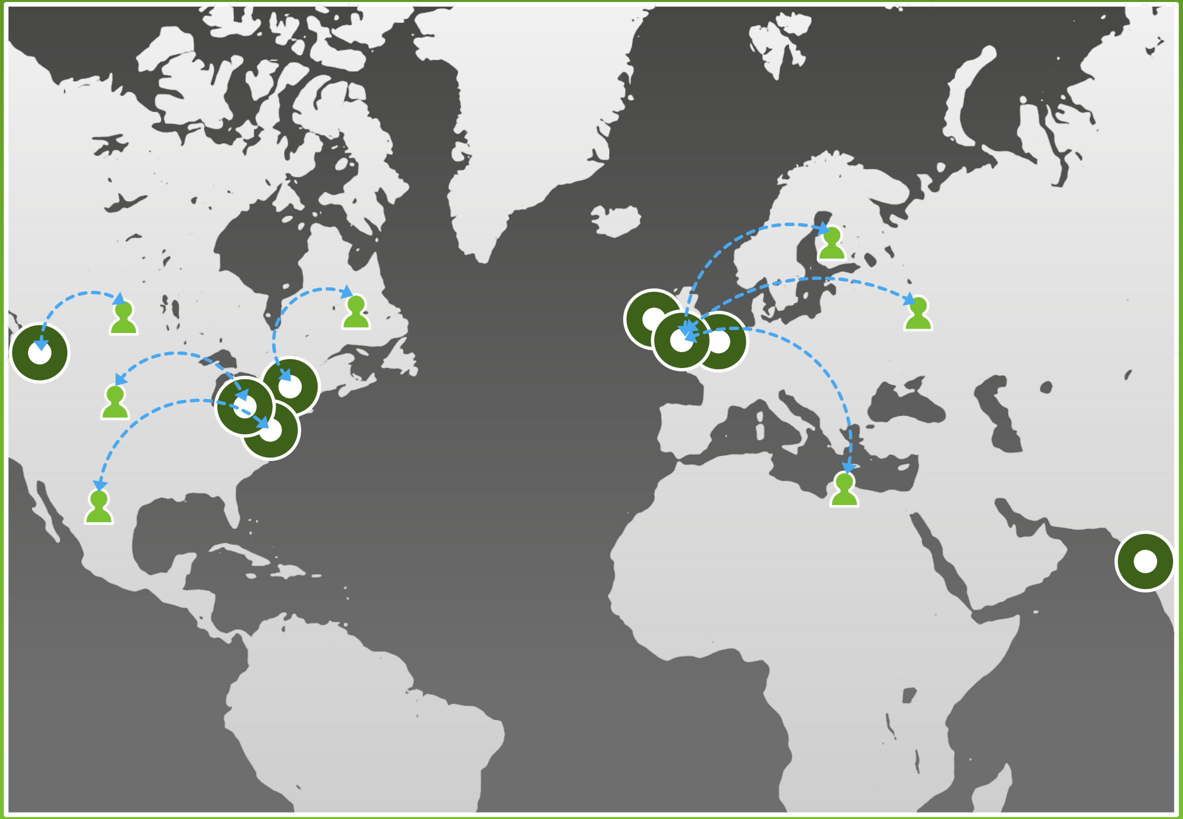
No single point of failure




Infrastructure

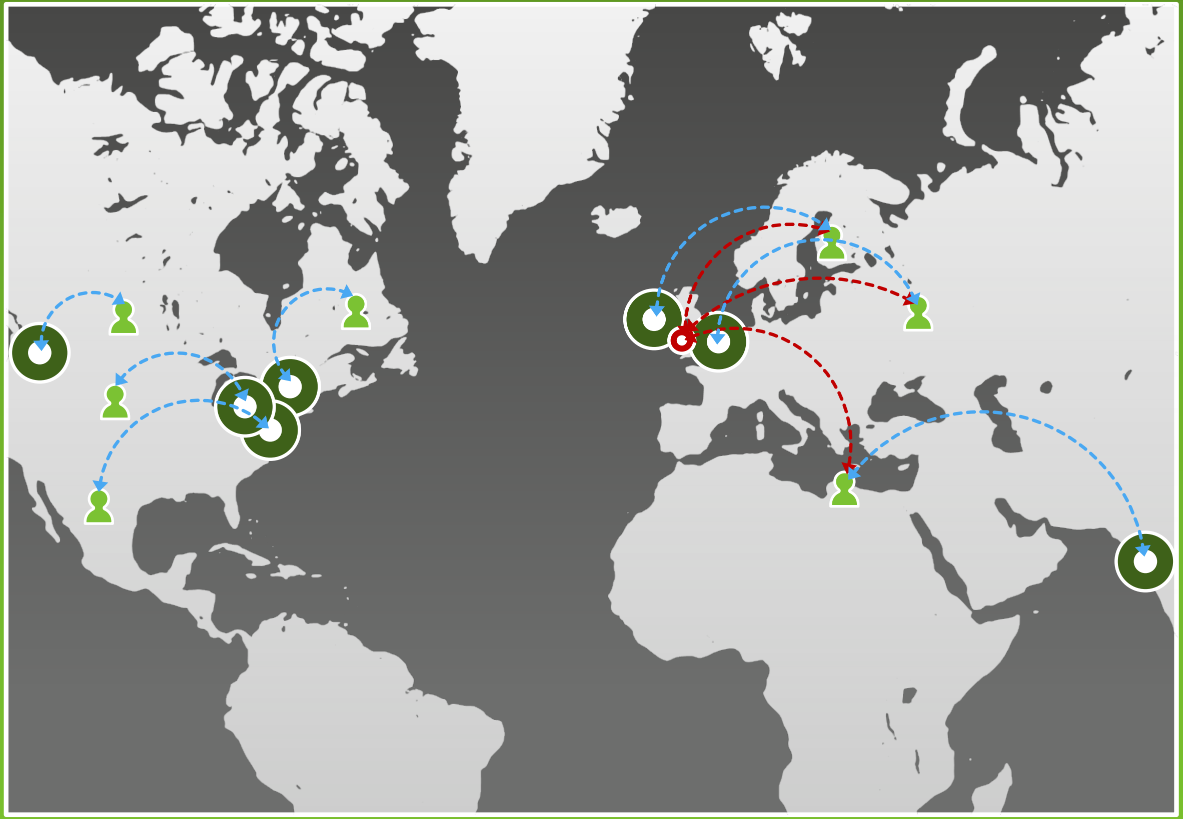
Switching and Interconnecting

Data replication
Traffic routing
Avoiding issues
Anti-entropy recovery



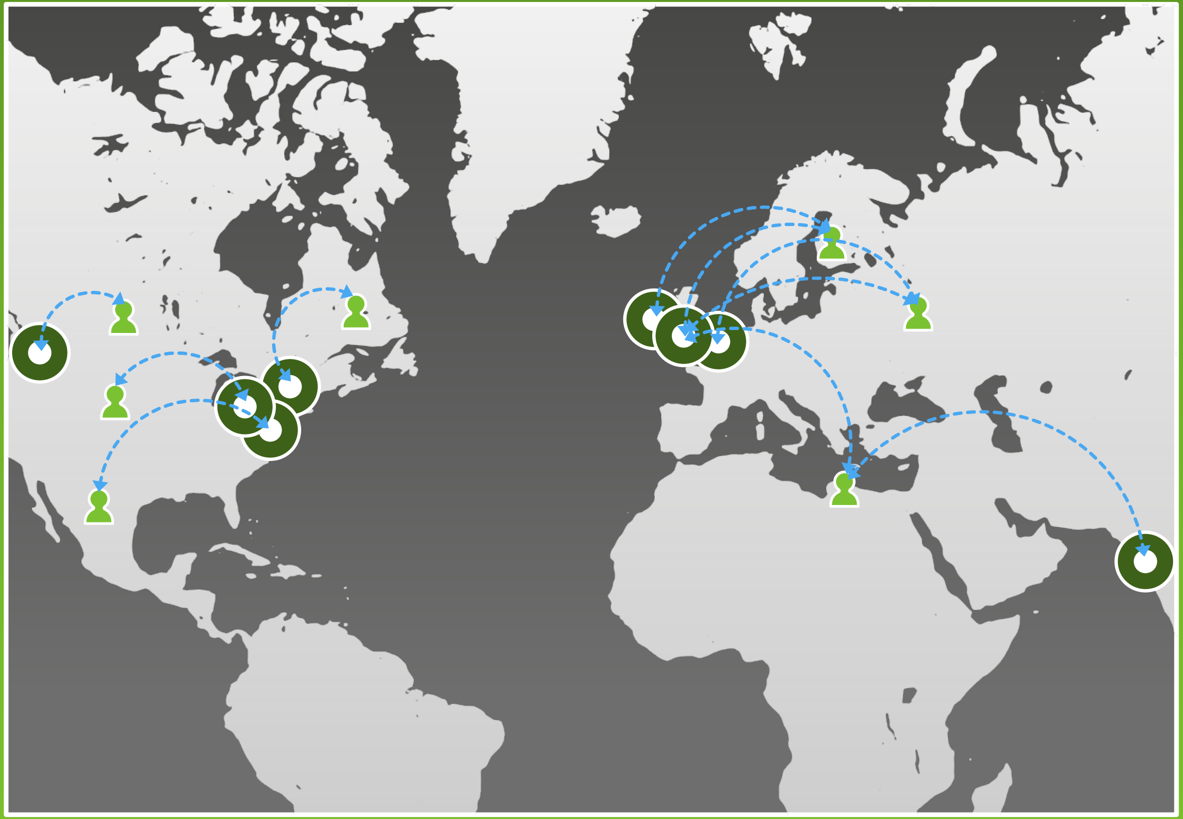
Switching and Interconnecting

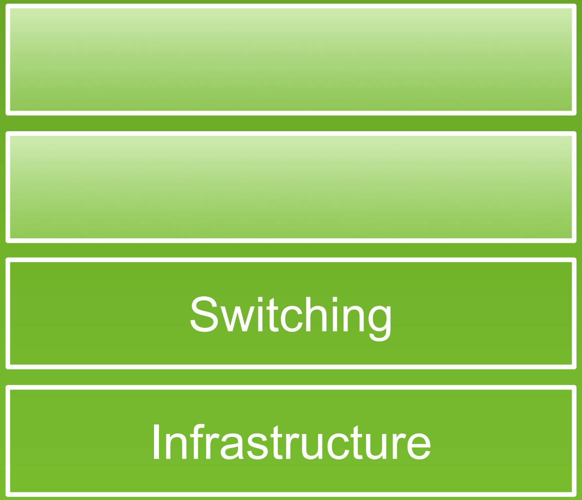
Data replication
Traffic routing
Avoiding issues
Anti-entropy recovery



Switching and Interconnecting

Data replication
Traffic routing
Avoiding issues
Anti-entropy recovery



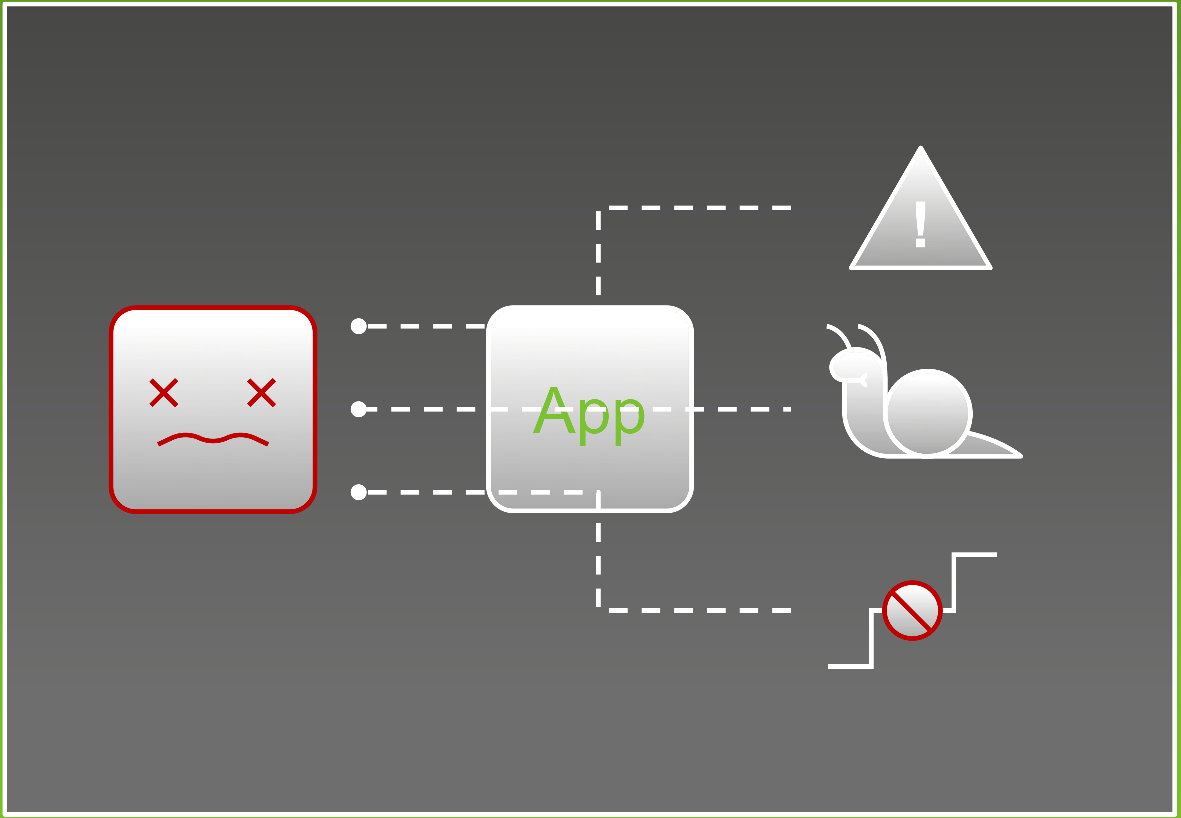

Switching
Infrastructure

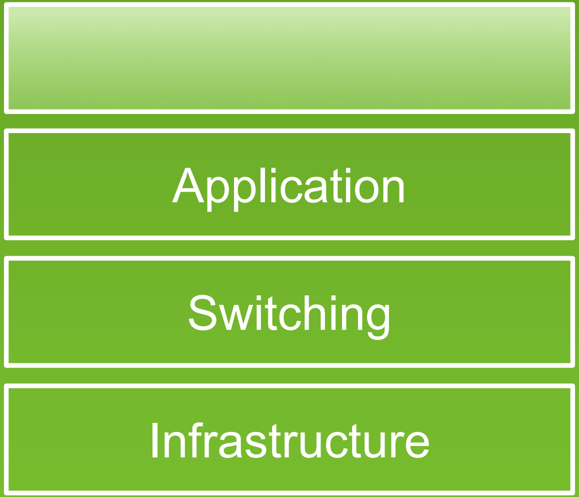
Application Failures

Error returns

Slow response

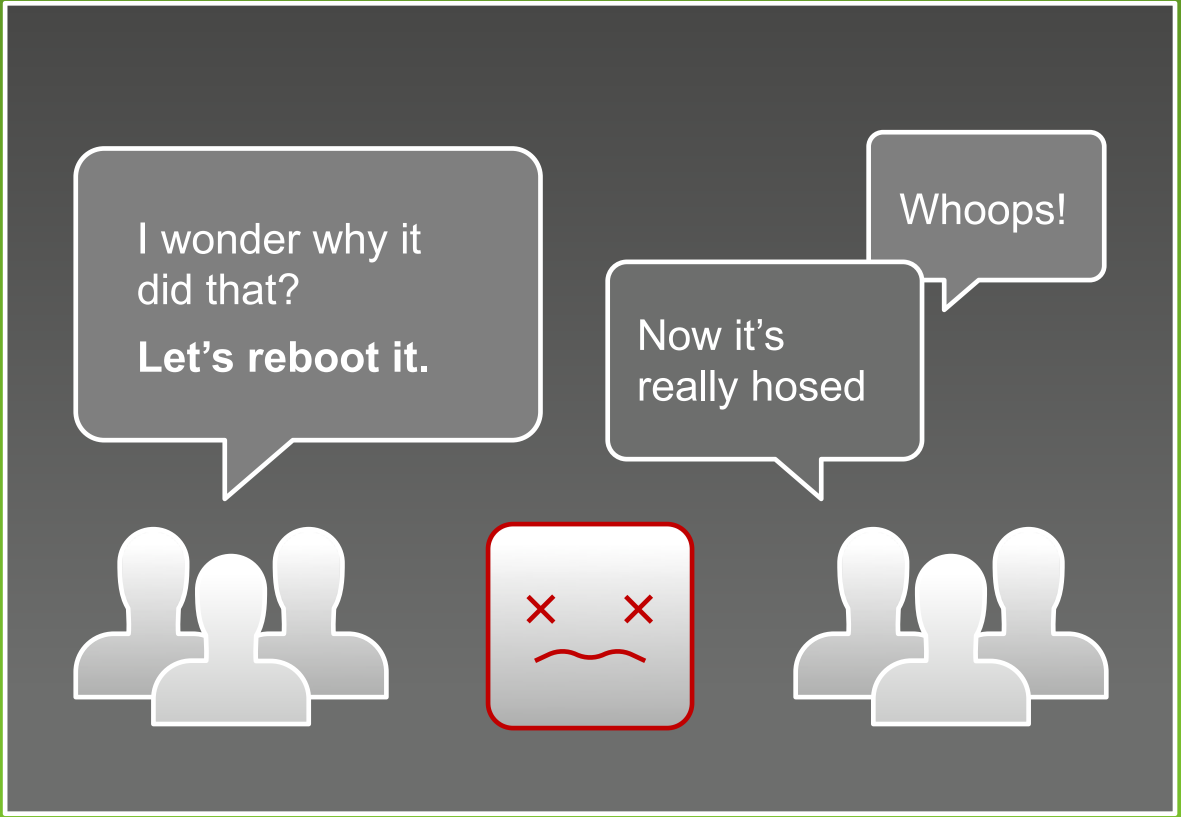
Network partition




Application
Switching
Infrastructure

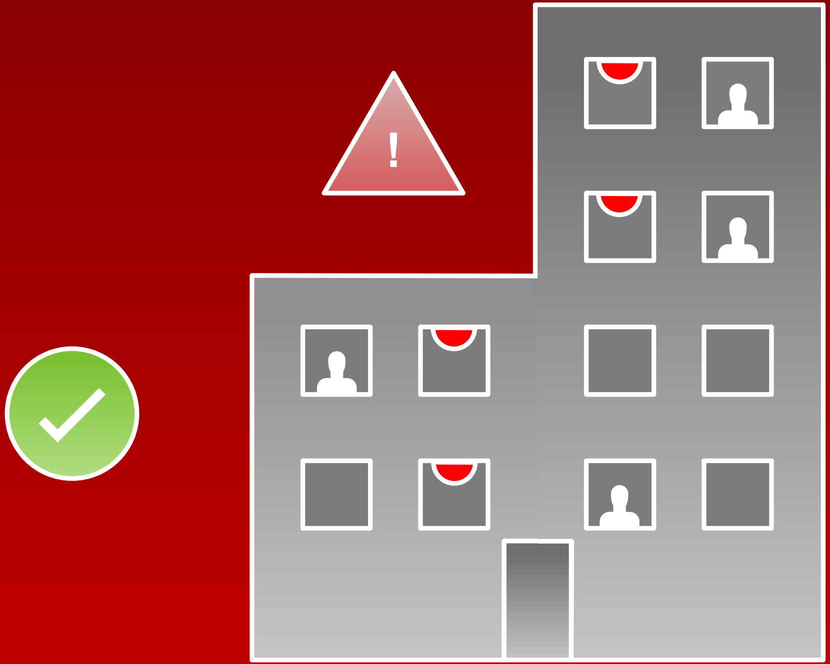
People

Unexpected application behavior often causes people to intervene and make the situation worse



People Training

A fire drill is a boring routine where we make everyone take the stairs and assemble in the parking lot



People Training

Fire drills save lives in the event of a real fire, because people are trained how to react

Who runs the “fire drill” for I.T.?

- People
- Application
- Switching
- Infrastructure

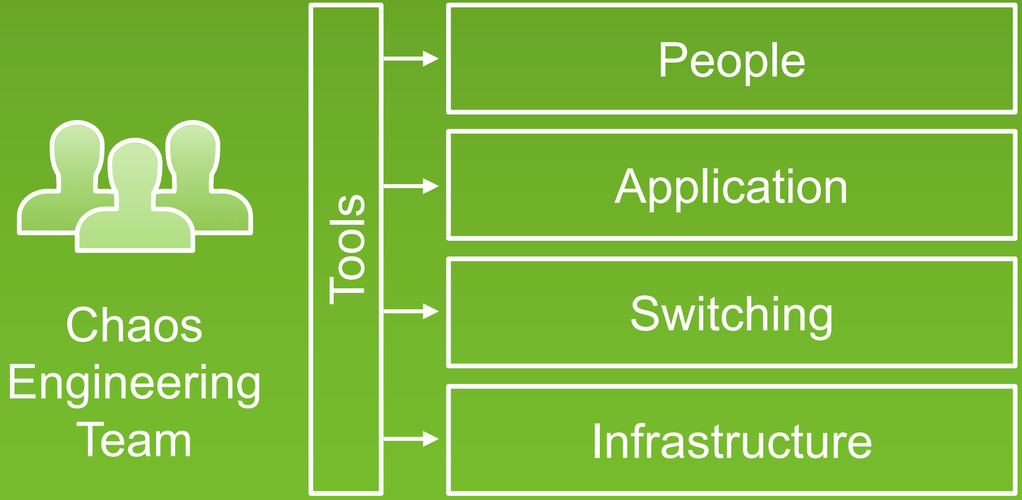
Chaos Engineering Team

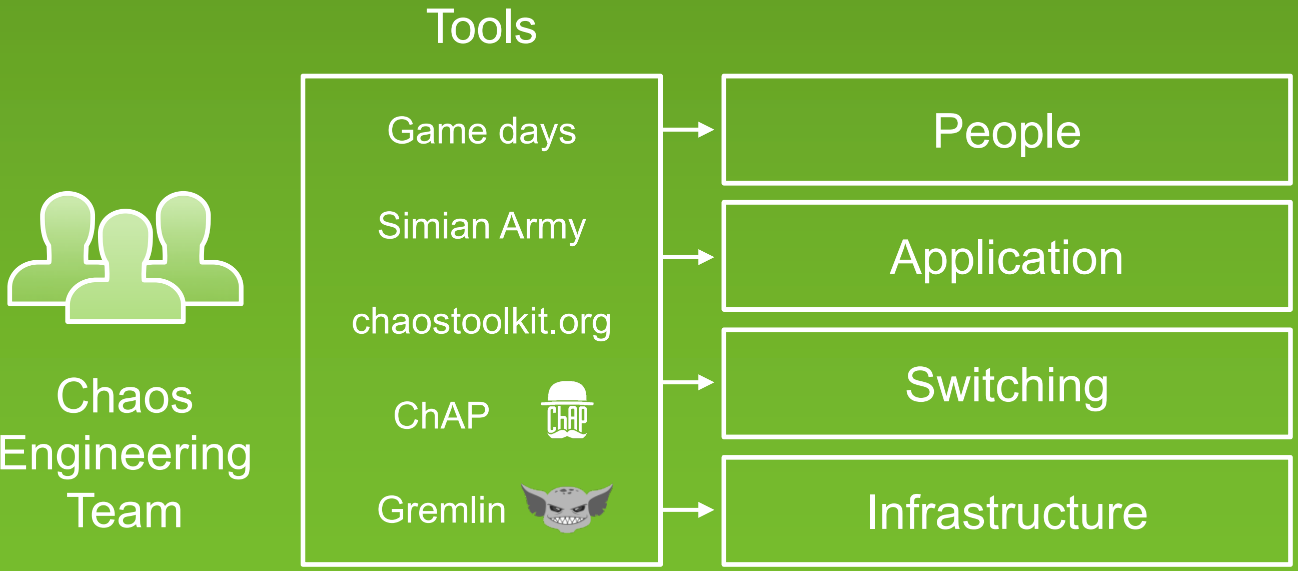
People

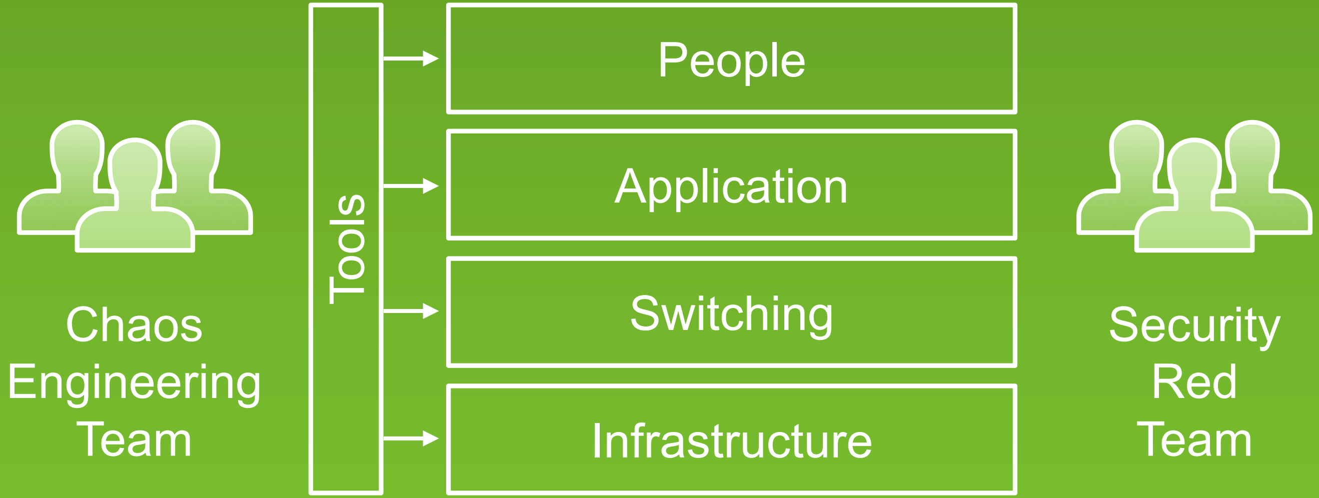
Application

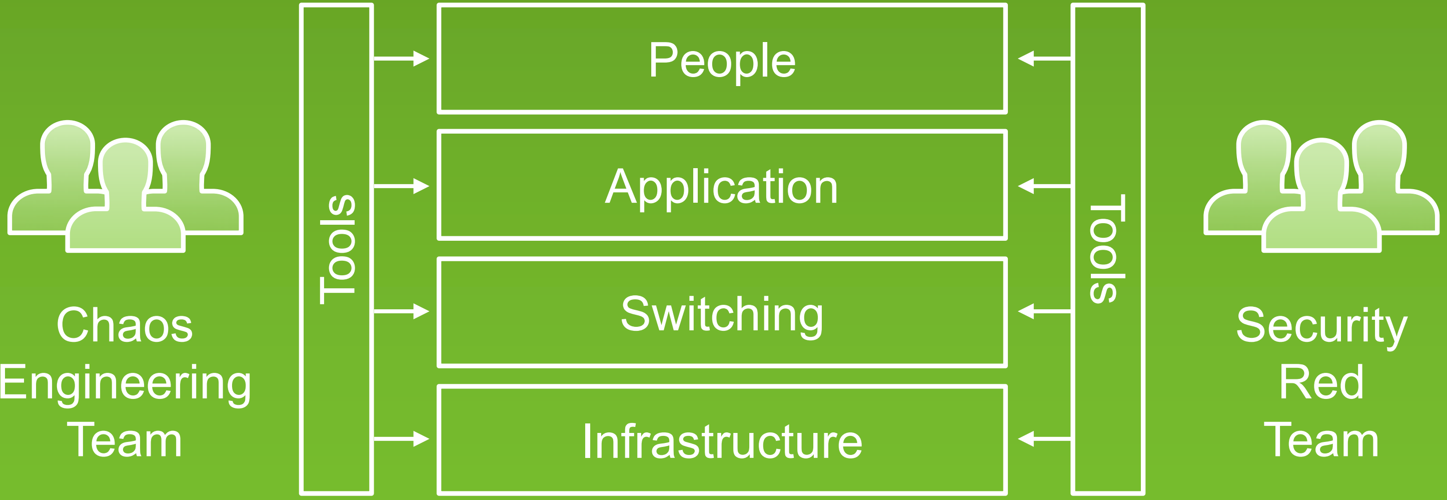
Switching

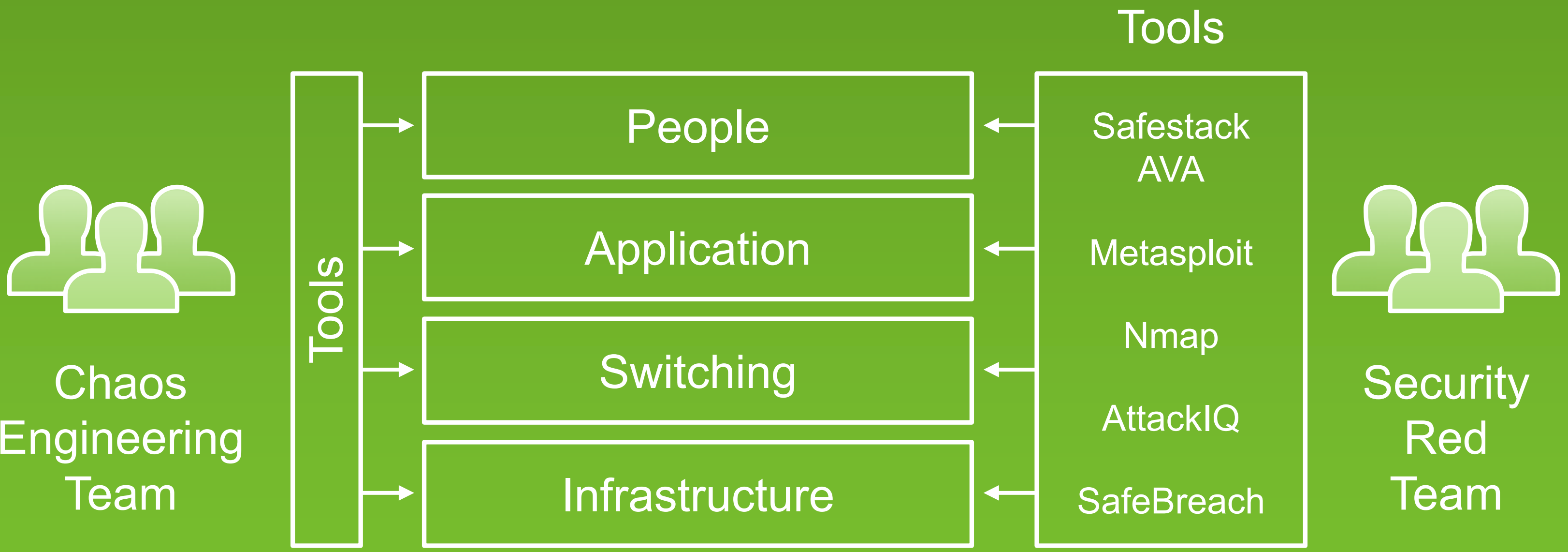
Infrastructure


Chaos Engineering Team
Tools
People
Application
Switching
Infrastructure


Chaos Engineering Team
Tools
Game days
Simian Army
chaostoolkit.org
ChAP
ChAP
Gremlin
People
Application
Switching
Infrastructure


Chaos Engineering Team
Tools
People
Application
Switching
Infrastructure
Security Red Team


Chaos
Engineering
Team
Tools
People
Application
Switching
Infrastructure
Tools
Security
Red
Team

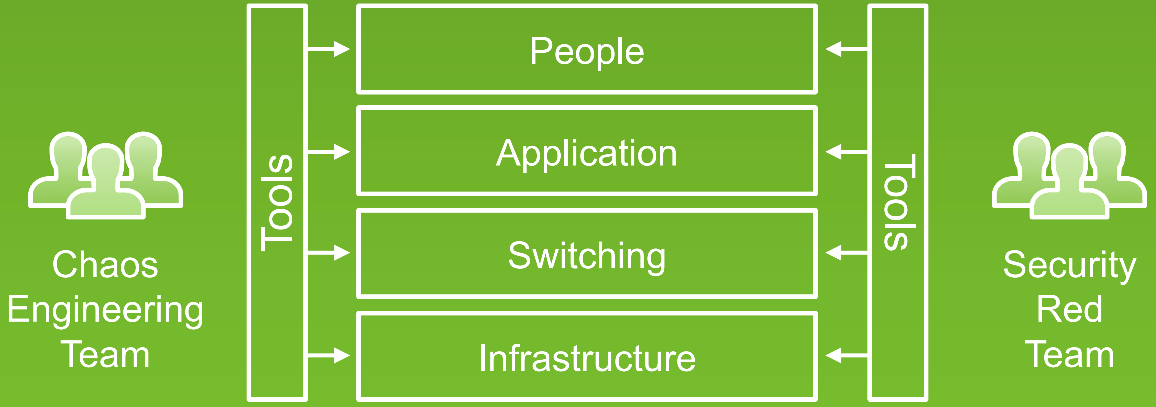

Chaos
Engineering
Team
Tools
People
Application
Switching
Infrastructure
Tools
Safestack
AVA
Metasploit
Nmap
AttackIQ
SafeBreach
Security
Red
Team

Chaos Architecture

Four layers

Two teams

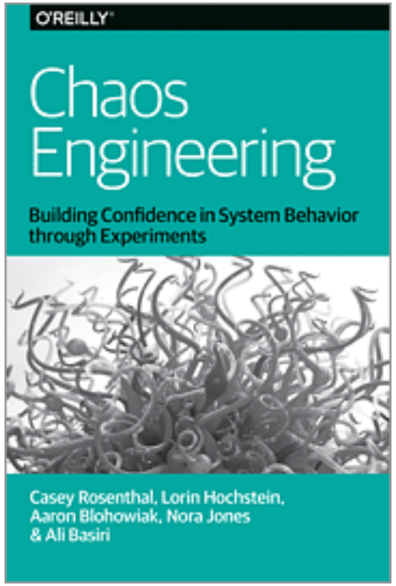
An attitude—
Break it to make it better



Cloud Trends

Thanks!

Adrian Cockcroft @adrianco
VP Cloud Architecture Strategy


O'REILLY

Chaos Engineering

Building Confidence in System Behavior through Experiments

Casey Rosenthal, Lorin Hochstein, Aaron Blohowiak, Nora Jones & Ali Basiri