CON208

AWS re:INVENT

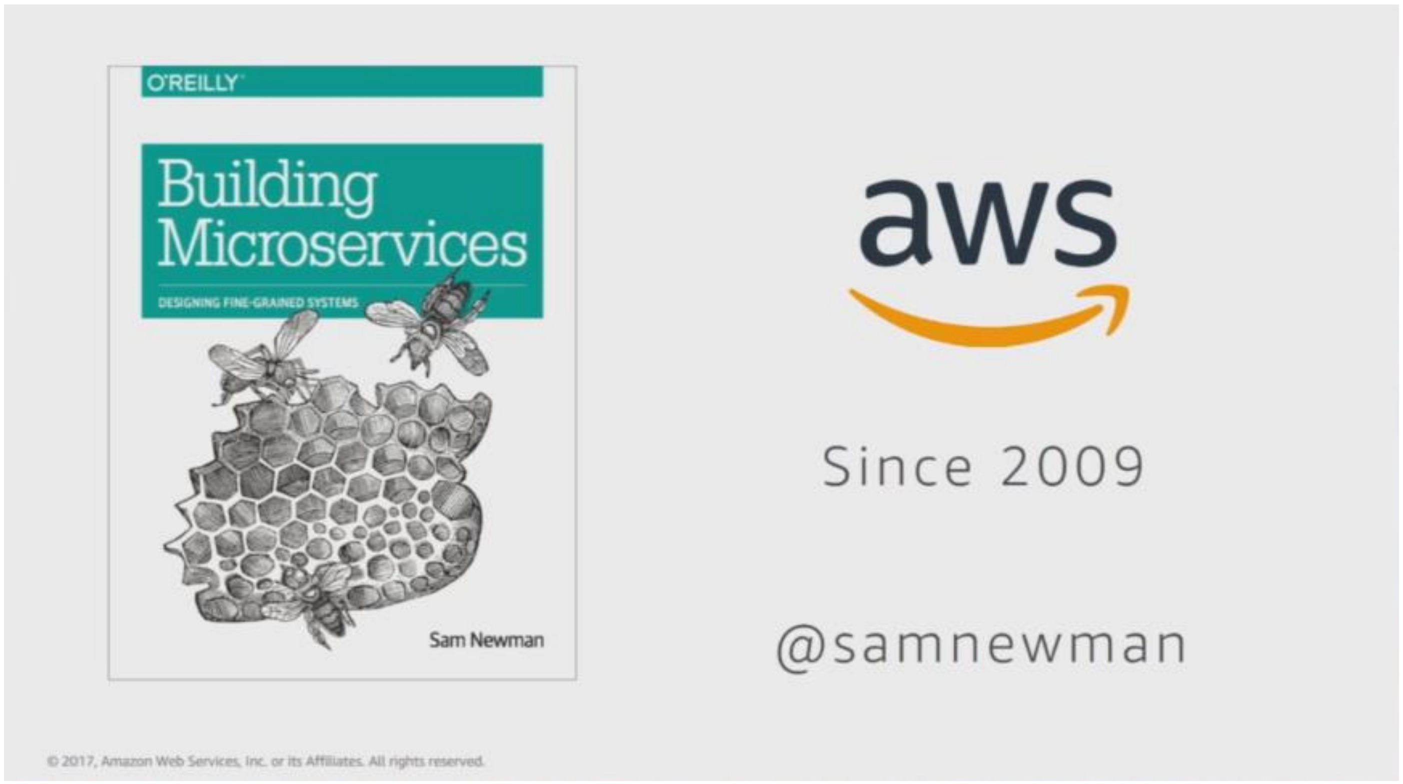
Building Microservices on AWS

Sam Newman

November 28, 2017

AWS re:Invent

Increasingly, organizations are turning to microservices to help them empower autonomous teams, letting them innovate and ship software faster than ever before. But implementing a microservices architecture comes with a number of new challenges that need to be dealt with. Chief among these finding an appropriate platform to help manage a growing number of independently deployable services. In this session, Sam Newman, author of Building Microservices and a renowned expert in microservices strategy, will discuss strategies for building scalable and robust microservices architectures, how to choose the right platform for building microservices, and common challenges and mistakes organizations make when they move to microservices architectures.



O'REILLY
Building Microservices
DESIGNING FINE-GRAINED SYSTEMS
Sam Newman

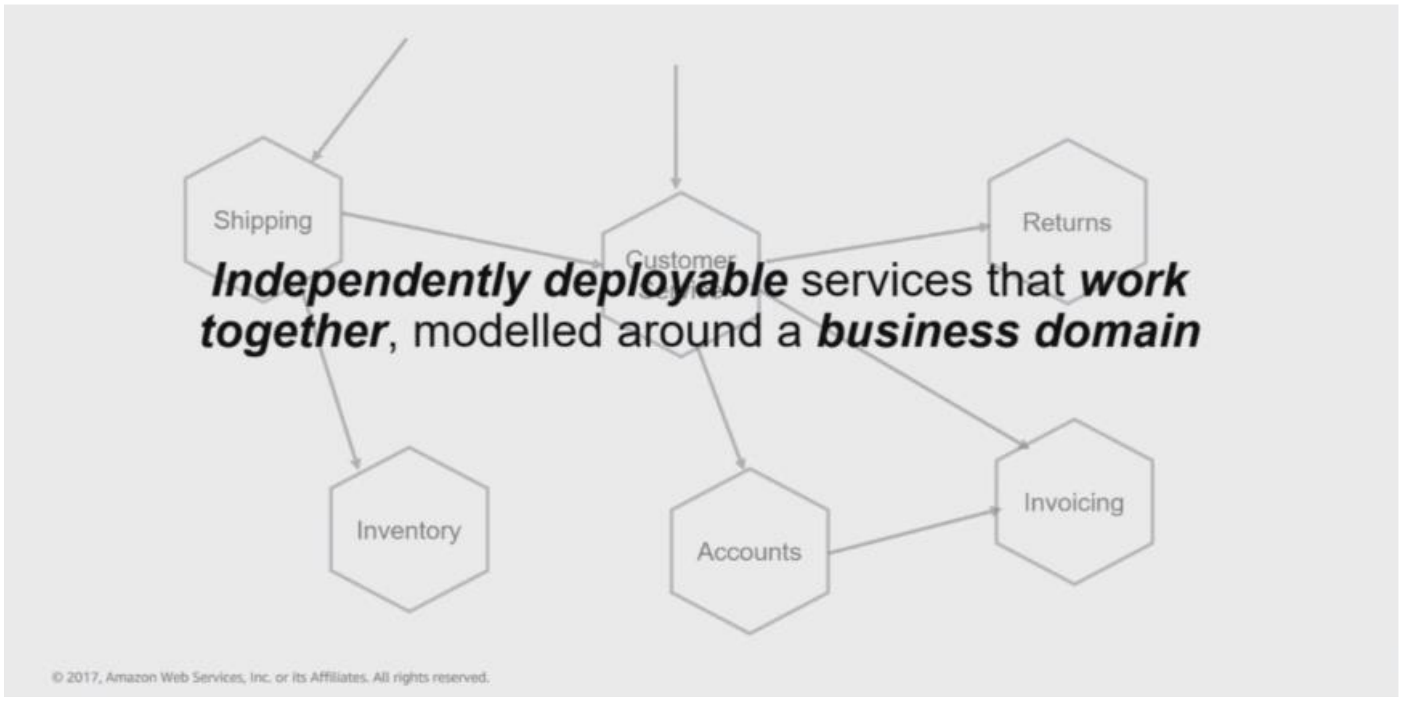
aws

Since 2009

@samnewman

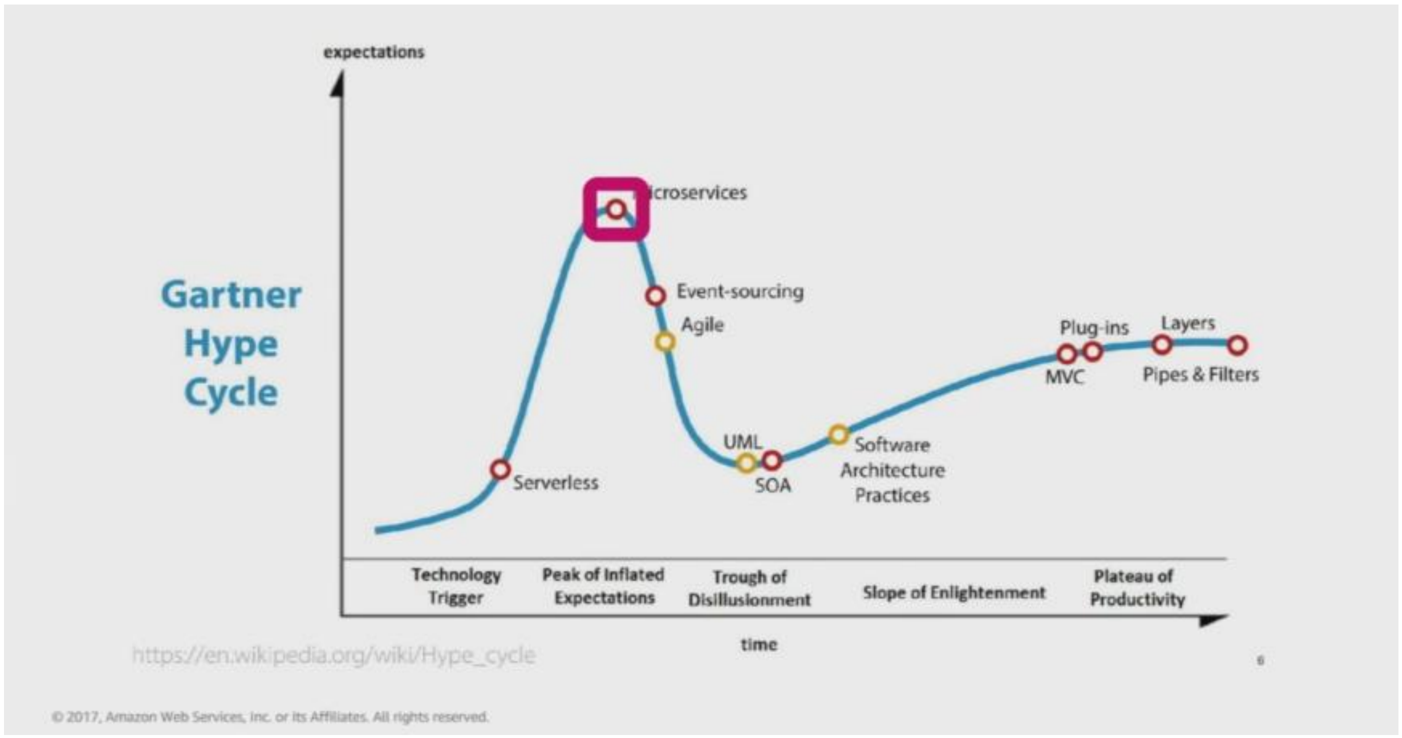
Sam Newman & Associates

I now run my own company doing consulting and training



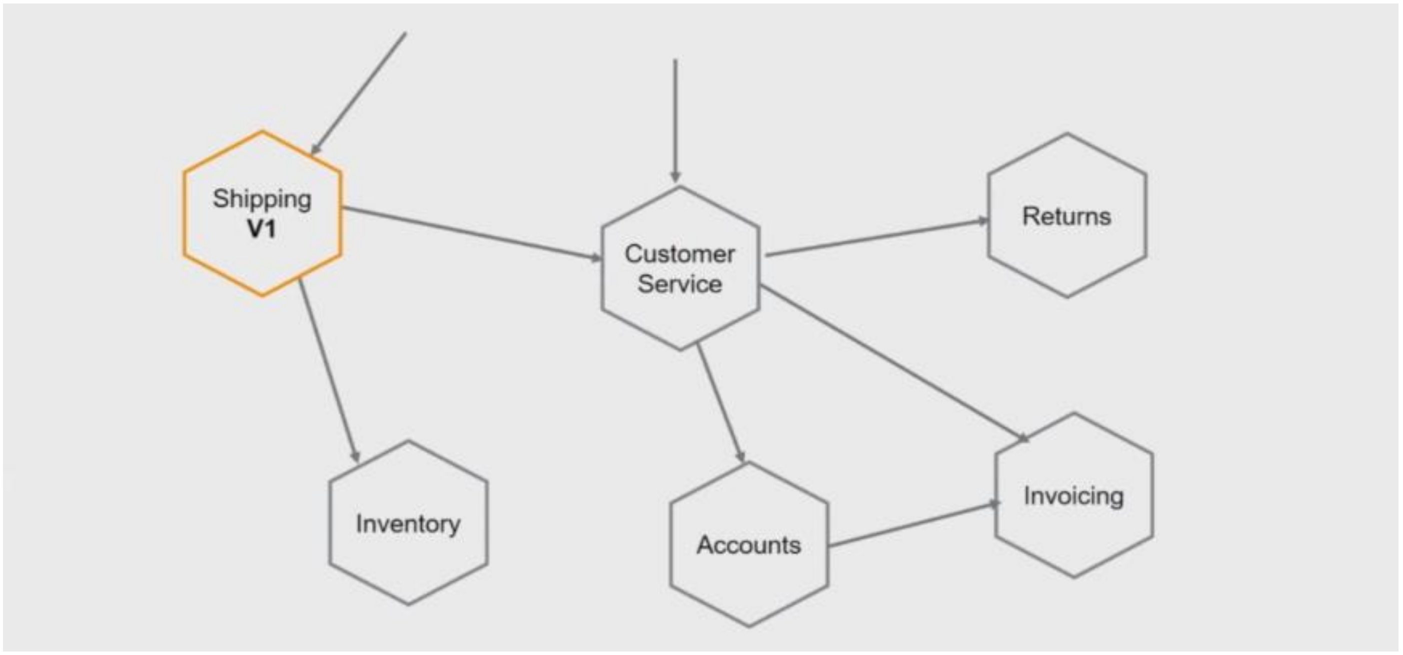
Independently deployable services that ***work together***, modelled around a ***business domain***

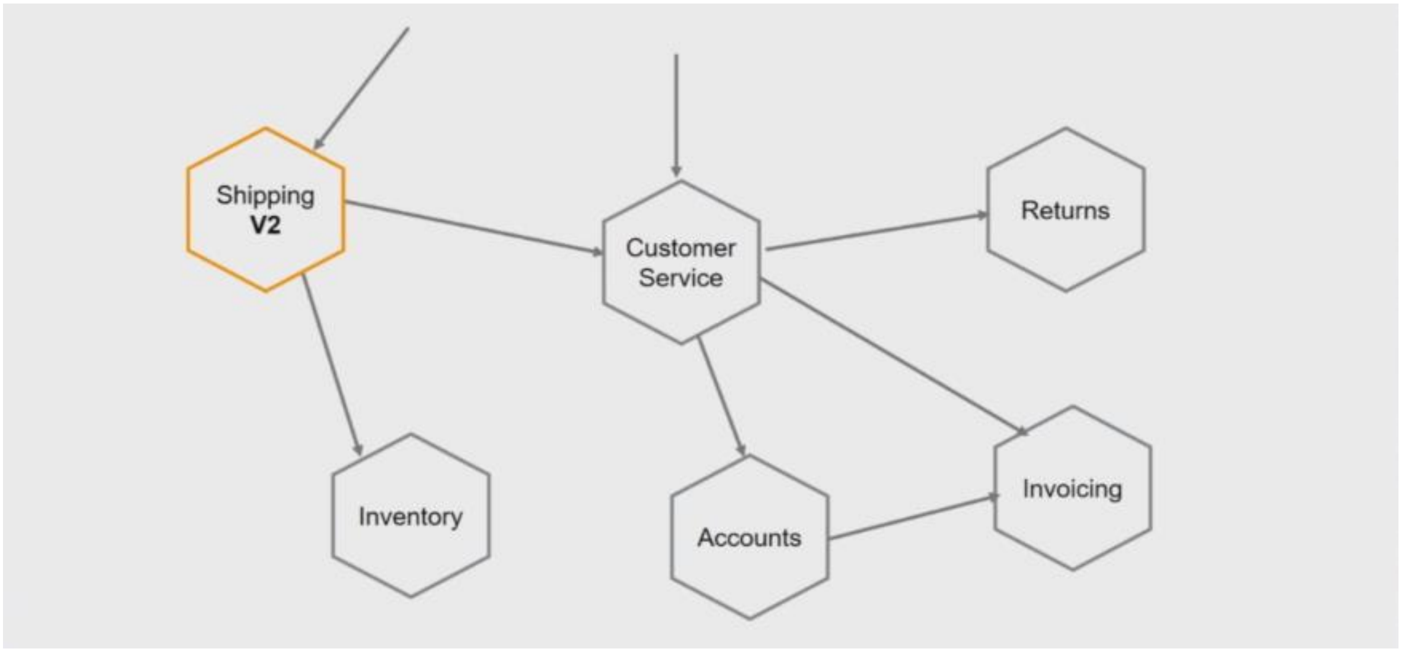
This is what a microservices architecture looks like



NETFLIX

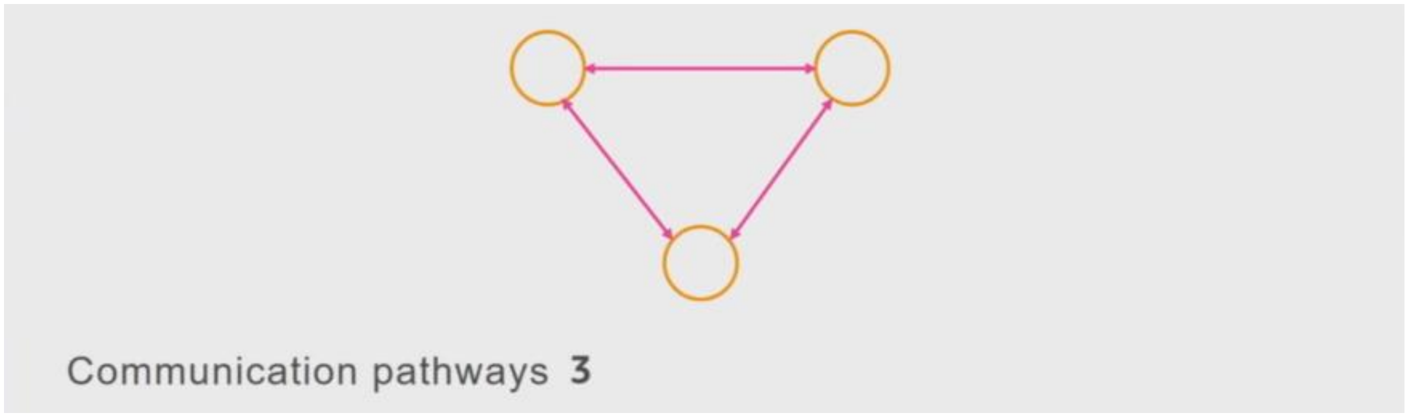
Independently deployable services that ***work together***, modelled around a ***business domain***

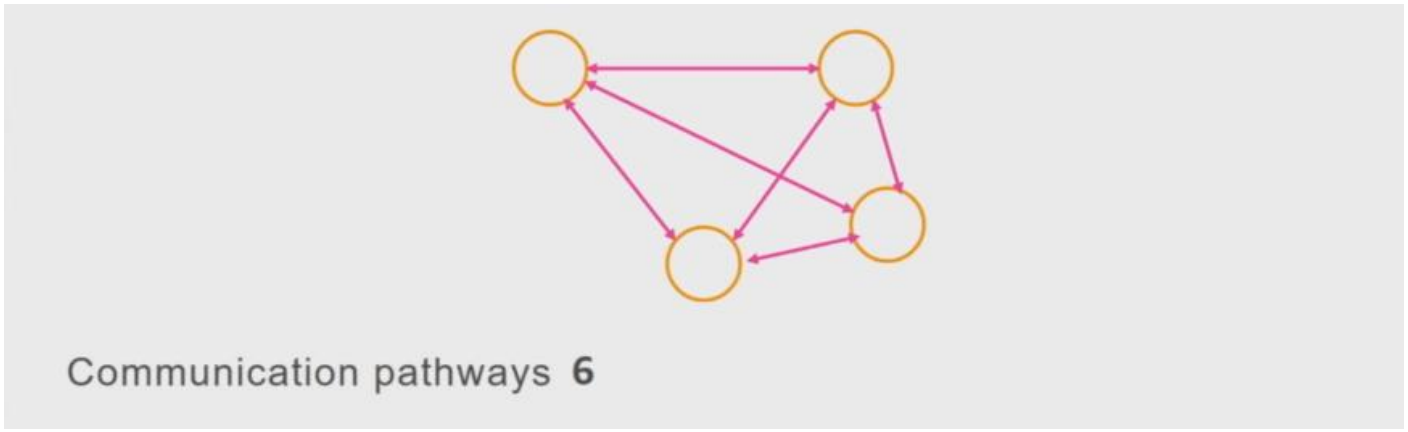


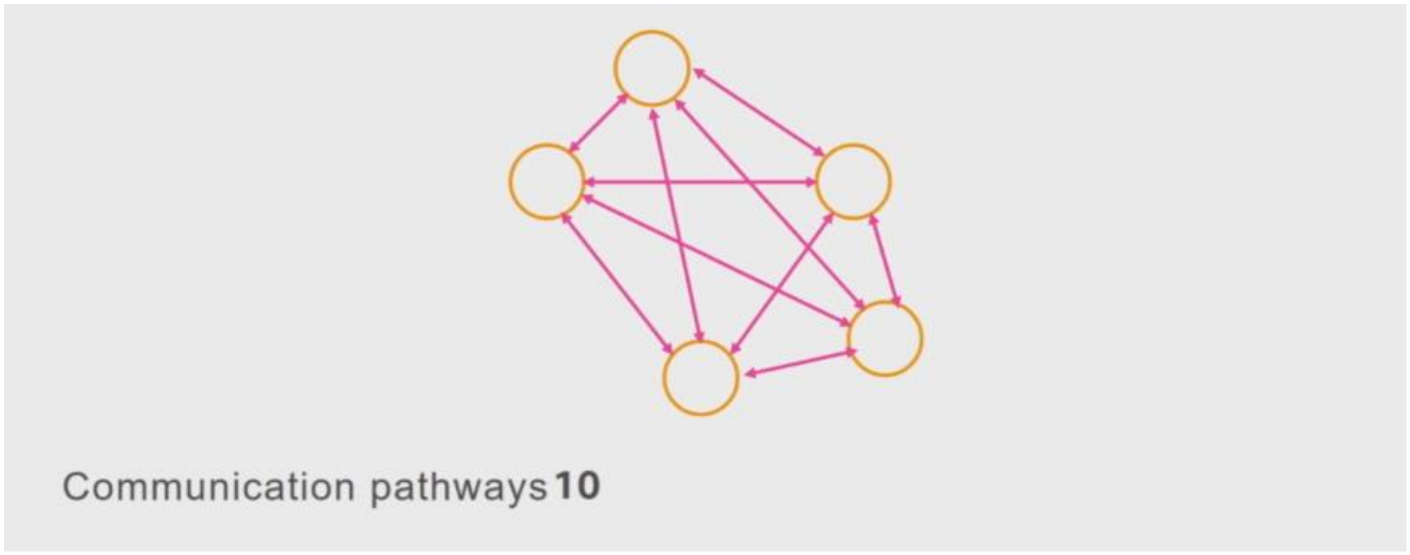


Autonomy

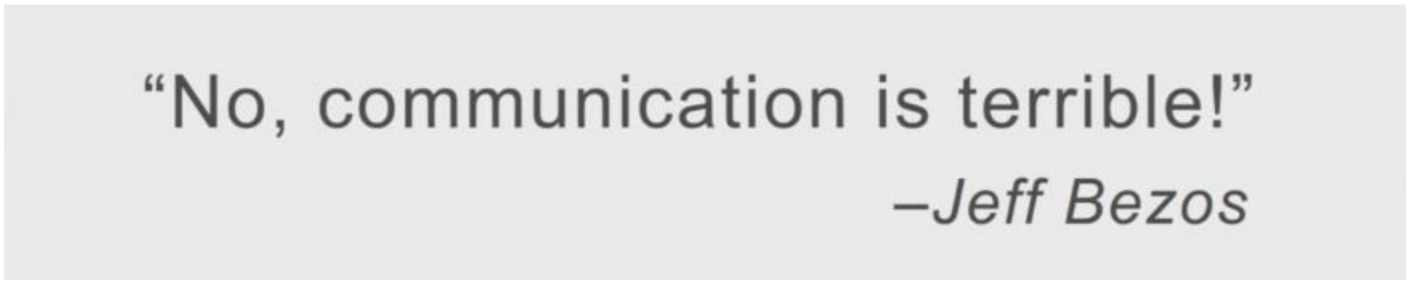
Autonomy of teams is all about reducing coordination so different groups can ship their code to production quickly.





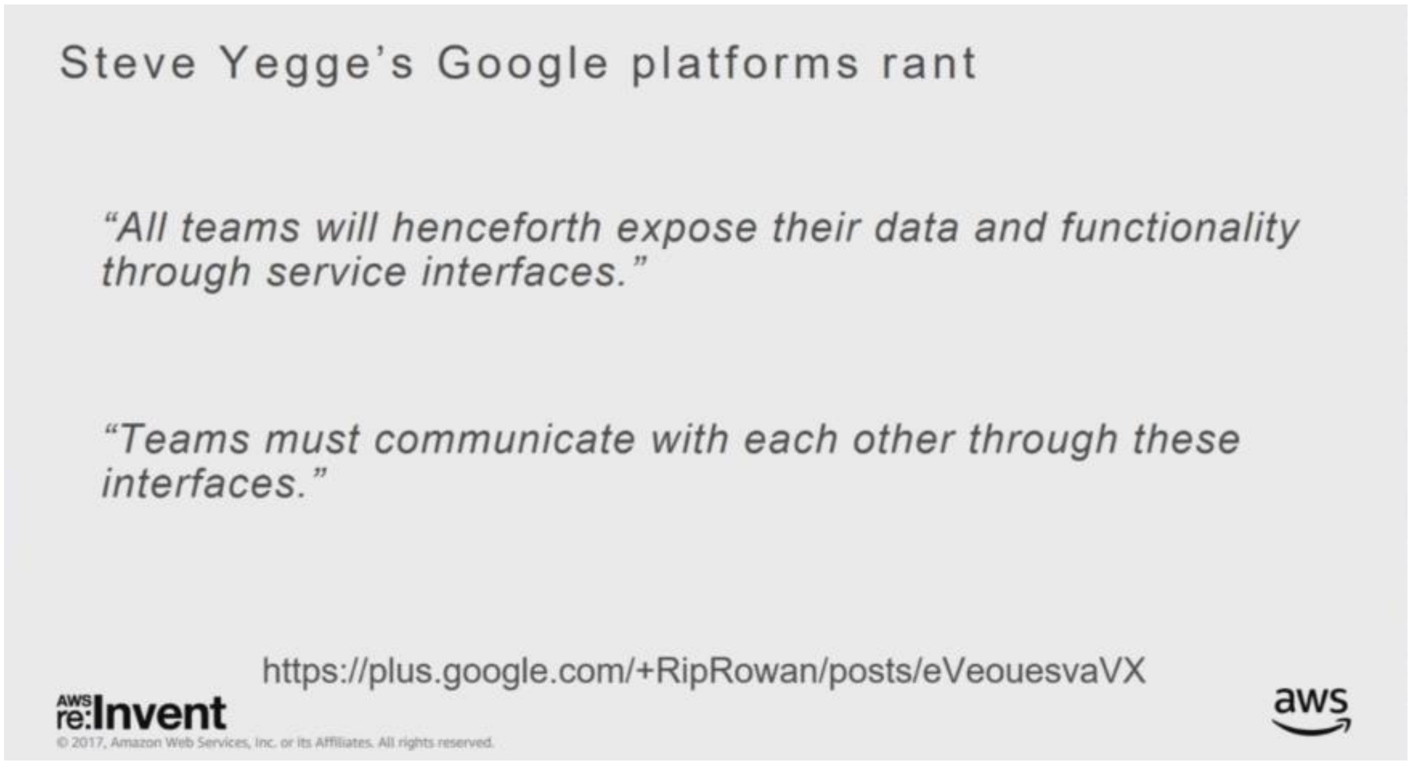


As we add more people to a team, coordination and communicating becomes a challenge.



"No, communication is terrible!"

–Jeff Bezos

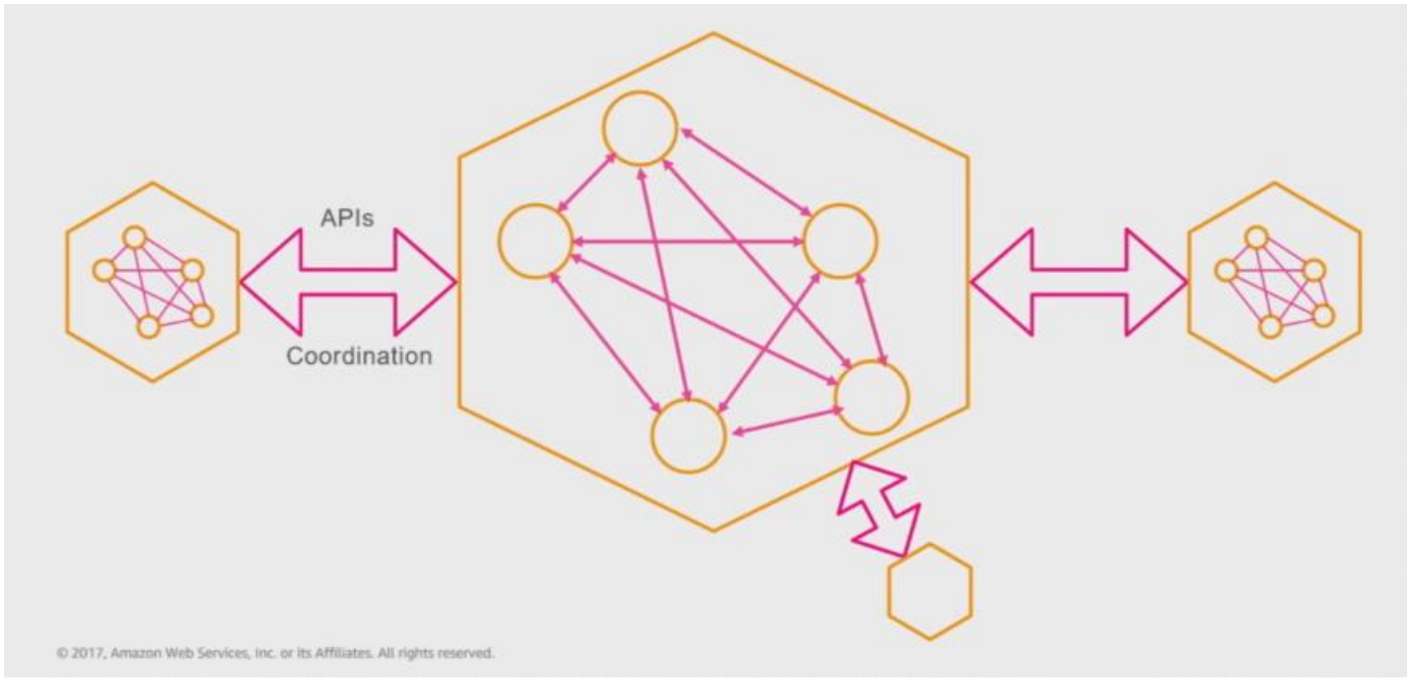


Steve Yegge's Google platforms rant

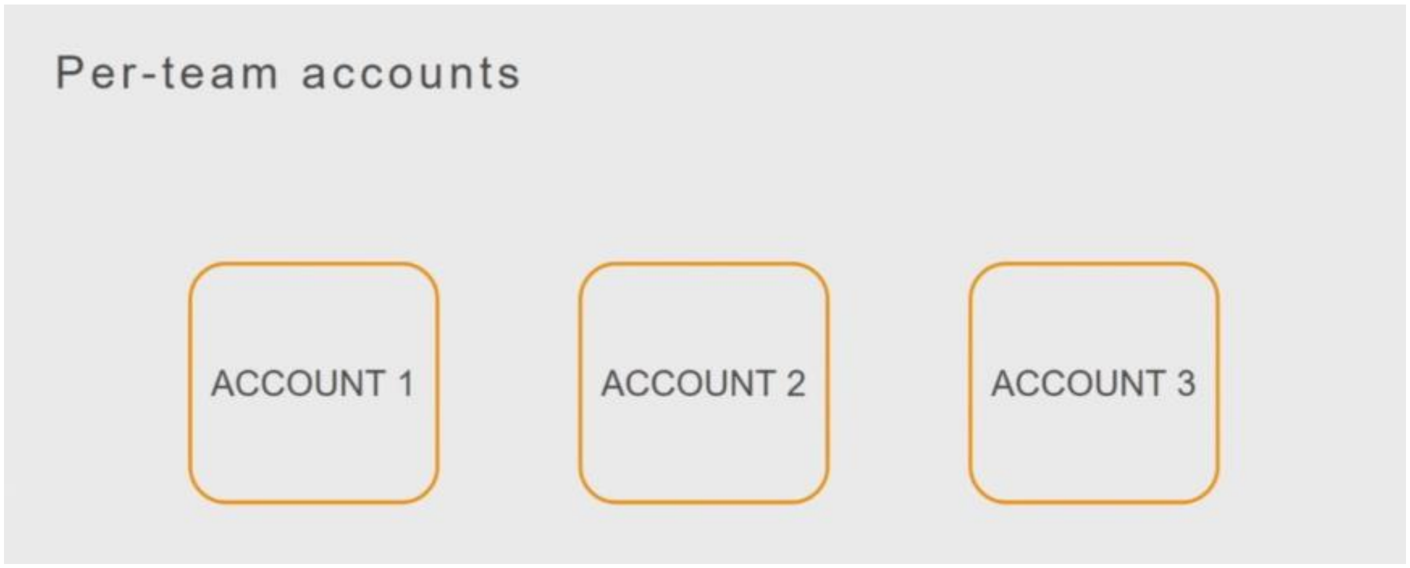
"All teams will henceforth expose their data and functionality through service interfaces."

"Teams must communicate with each other through these interfaces."

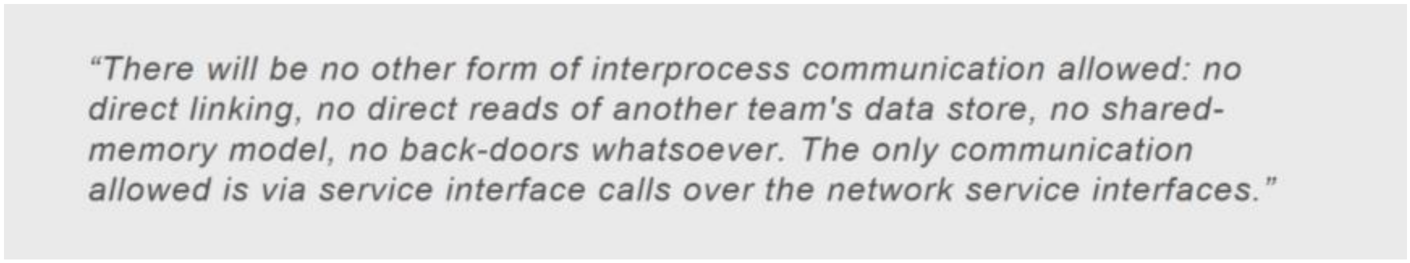
https://plus.google.com/+RipRowan/posts/eVeouesvaVX



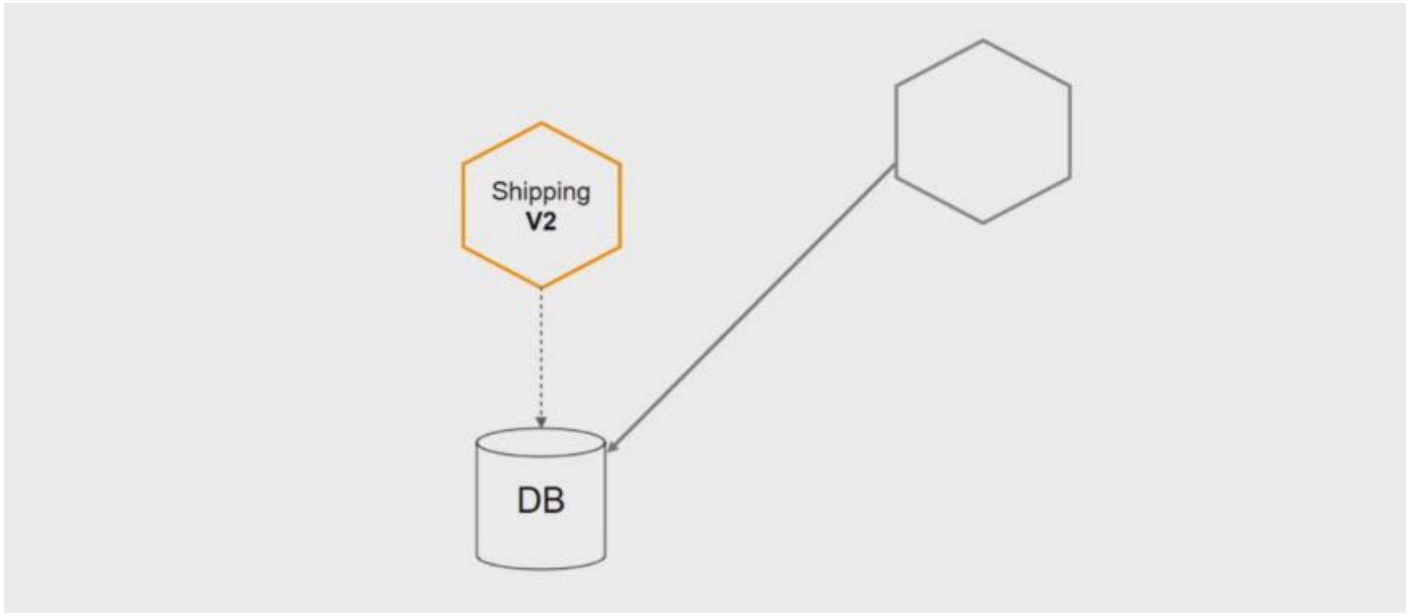
These are coarse grain coordination between services around their API points,

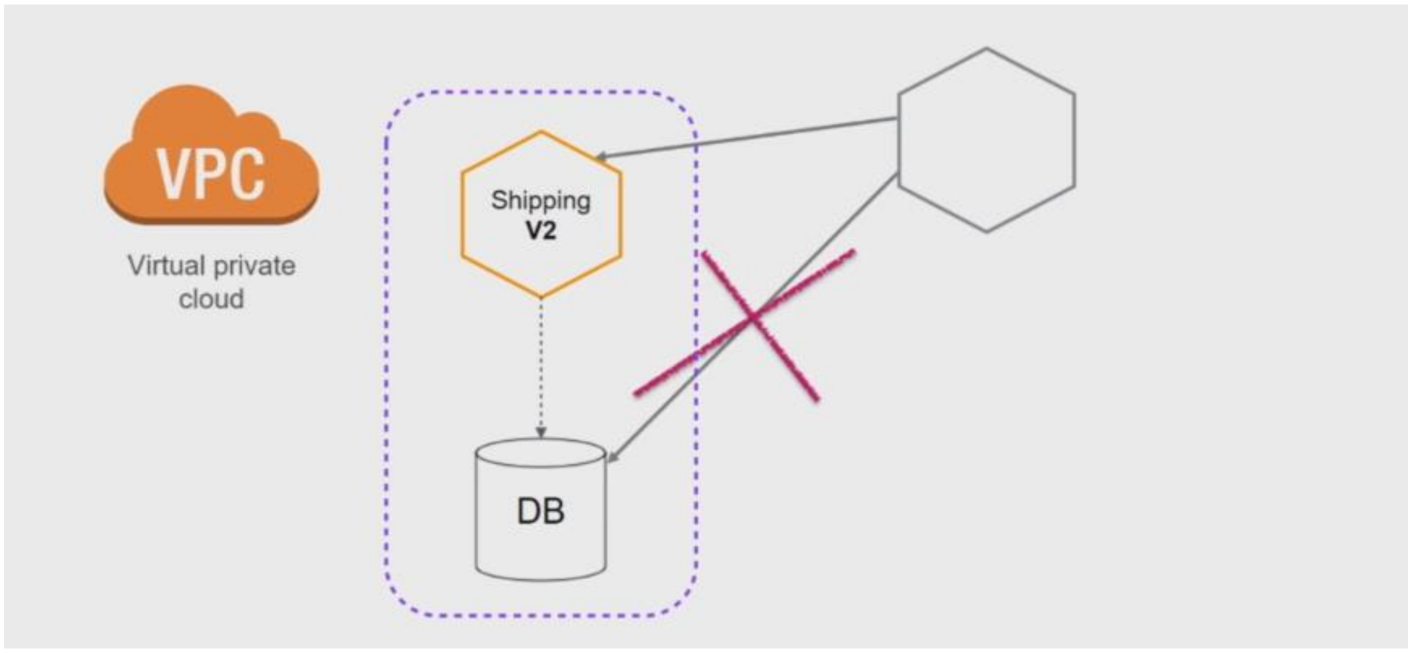


Per-team accounts

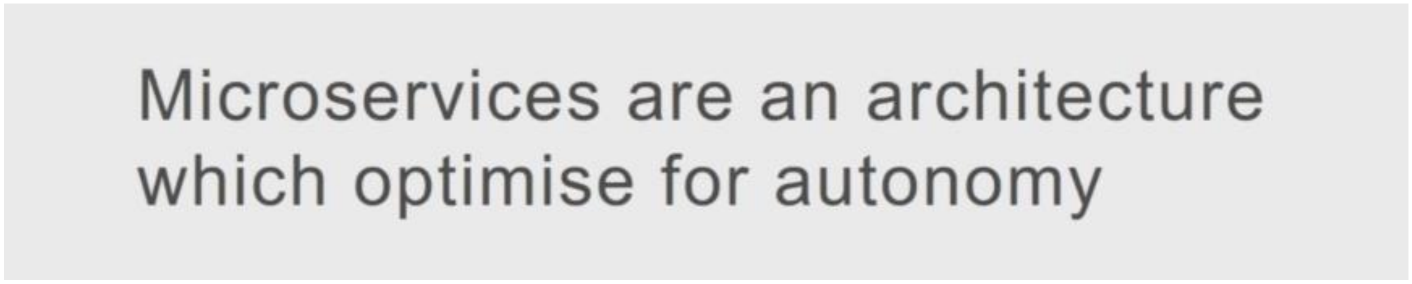


"There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network service interfaces."

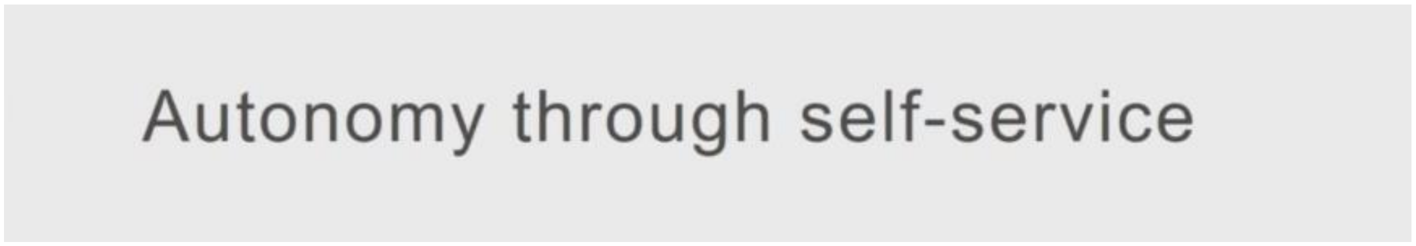




We can start thinking about our services like black boxes that have service interfaces that other services and users can interact with, the service's data store becomes a mere internal abstraction/implementation. We can use the VPC to turn off access to all internal parts of the service except the service interfaces to enforce the constraint as above.

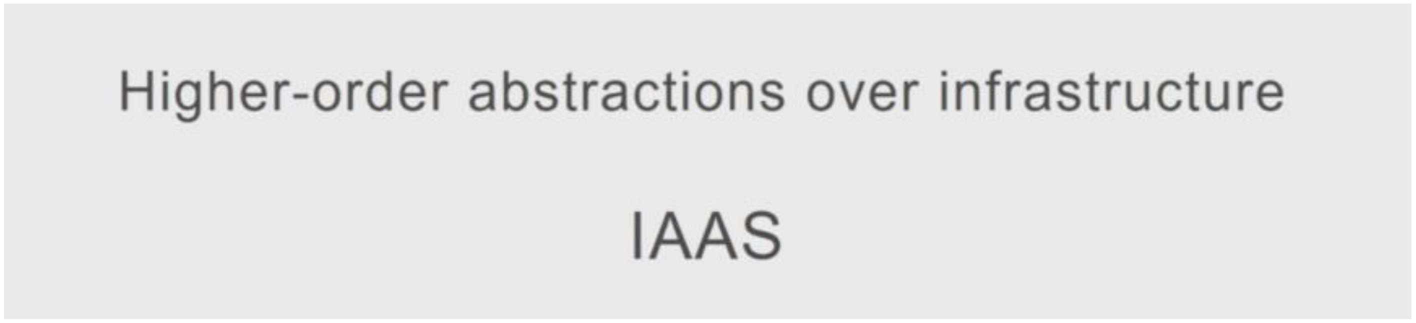


Microservices are an architecture which optimise for autonomy



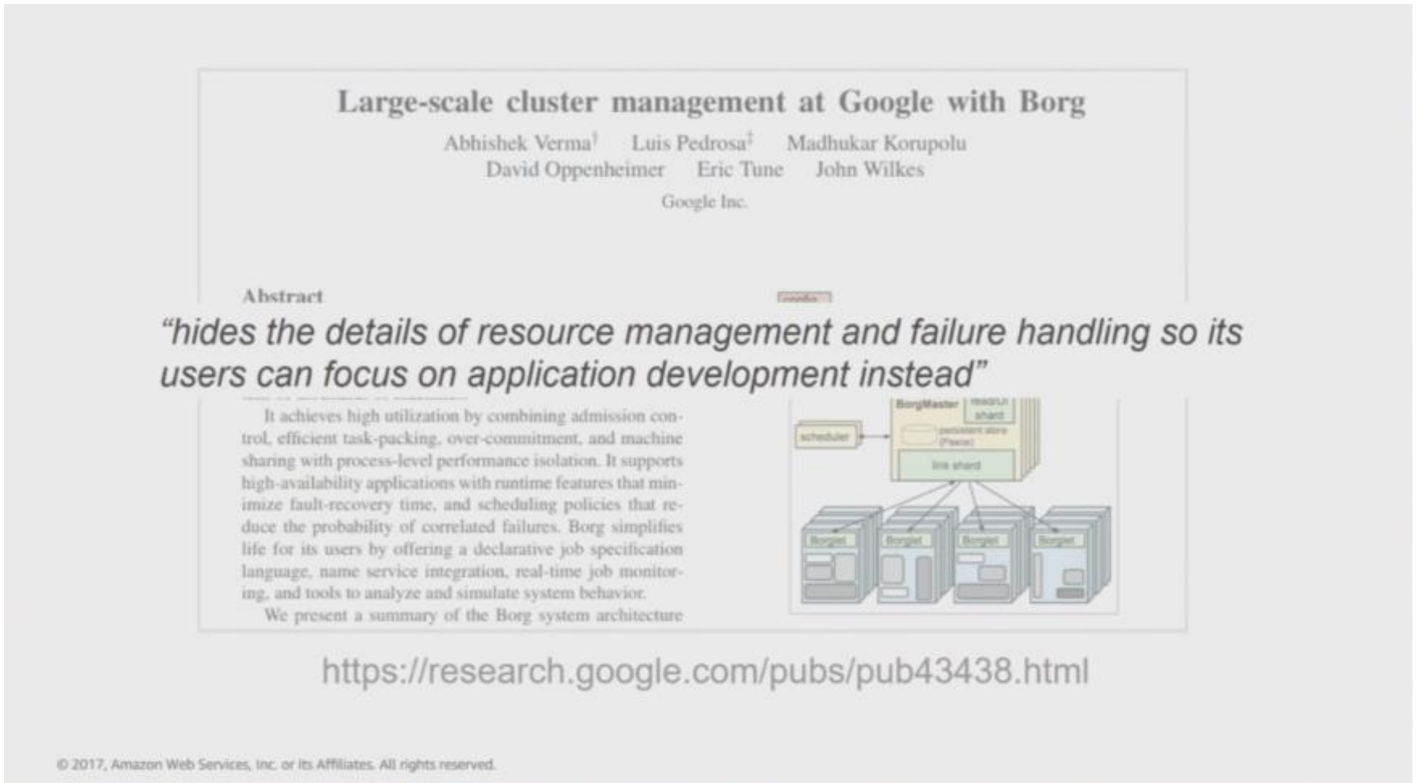
Autonomy through self-service

Small team autonomy in Amazon required abstractions over compute, storage, and the network requirements that led to the AWS product suite.



Higher-order abstractions over infrastructure

IAAS



Large-scale cluster management at Google with Borg

Abhishek Verma[†] Luis Pedrosa[‡] Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes

Google Inc.

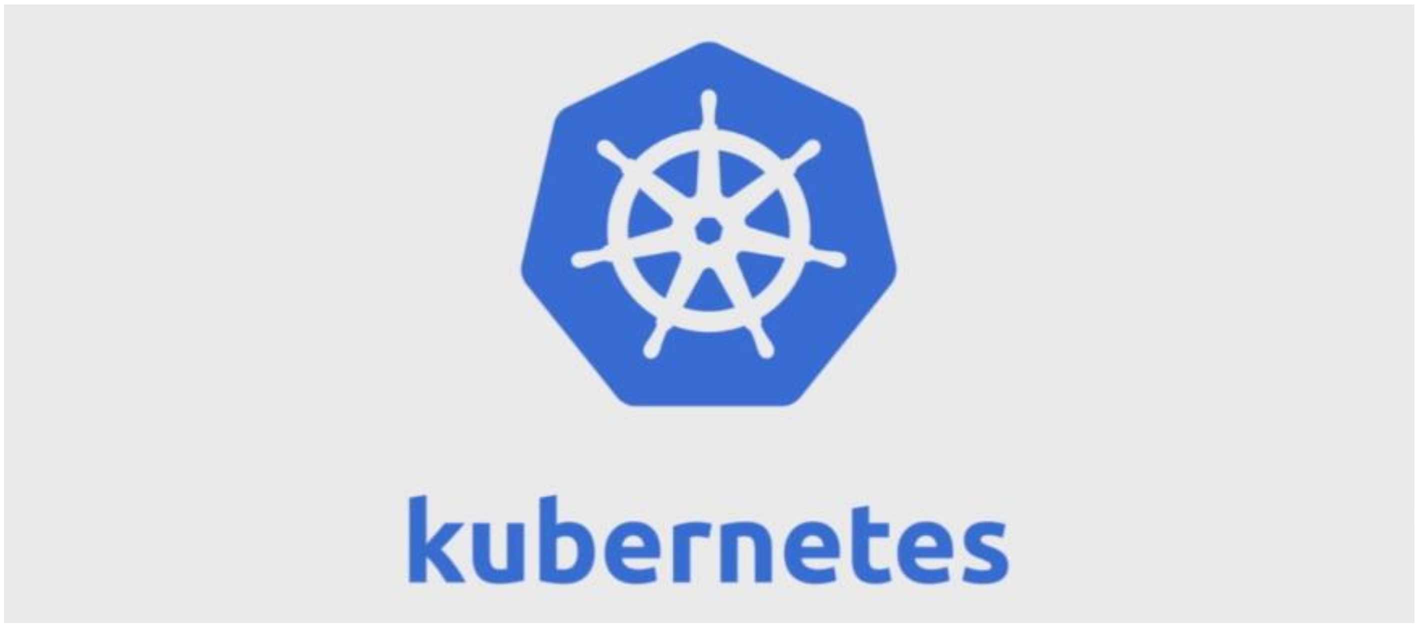
Abstract

"hides the details of resource management and failure handling so its users can focus on application development instead"

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture

https://research.google.com/pubs/pub43438.html



kubernetes

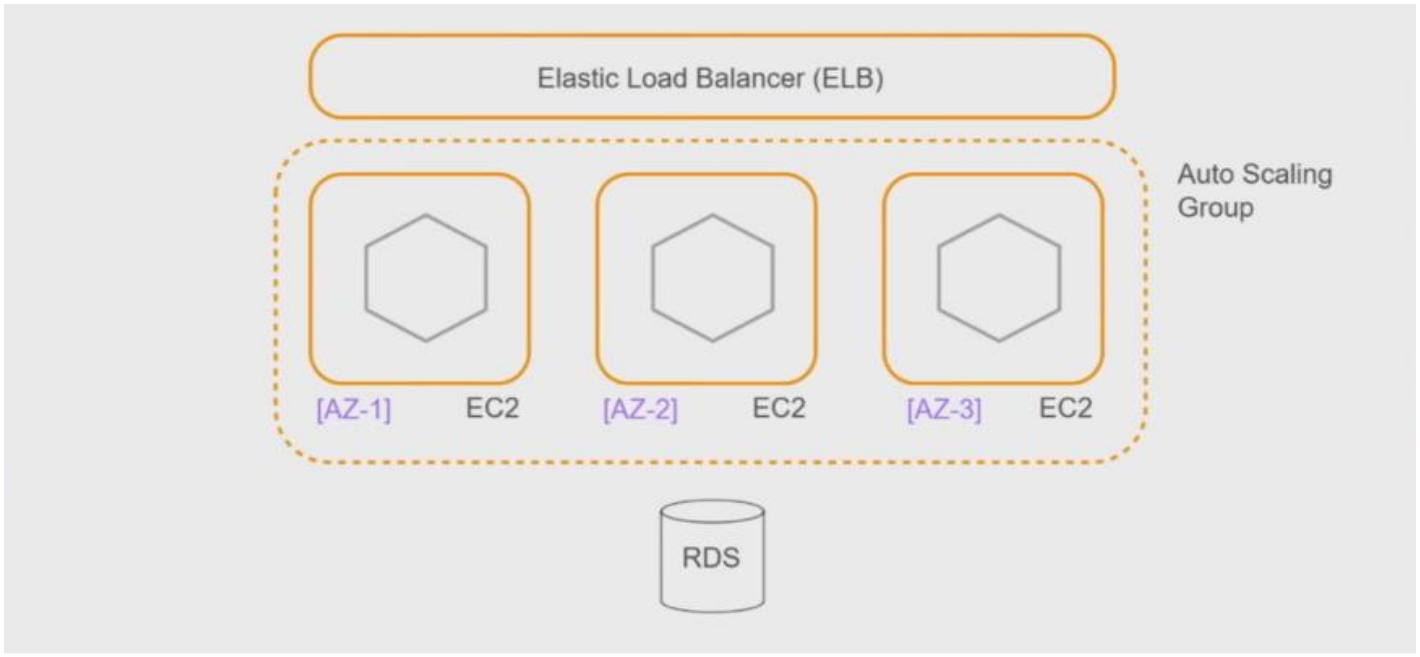
aws

NETFLIX

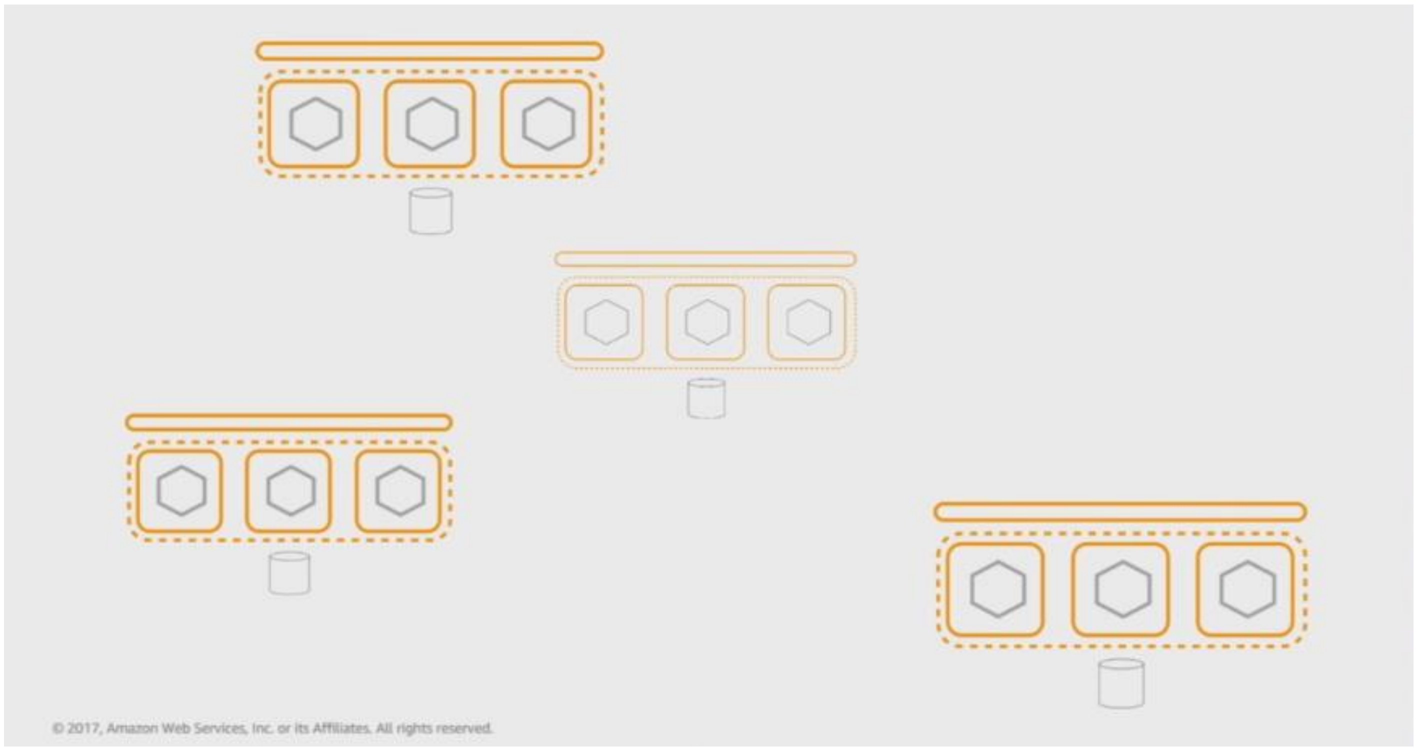
lyft

slack

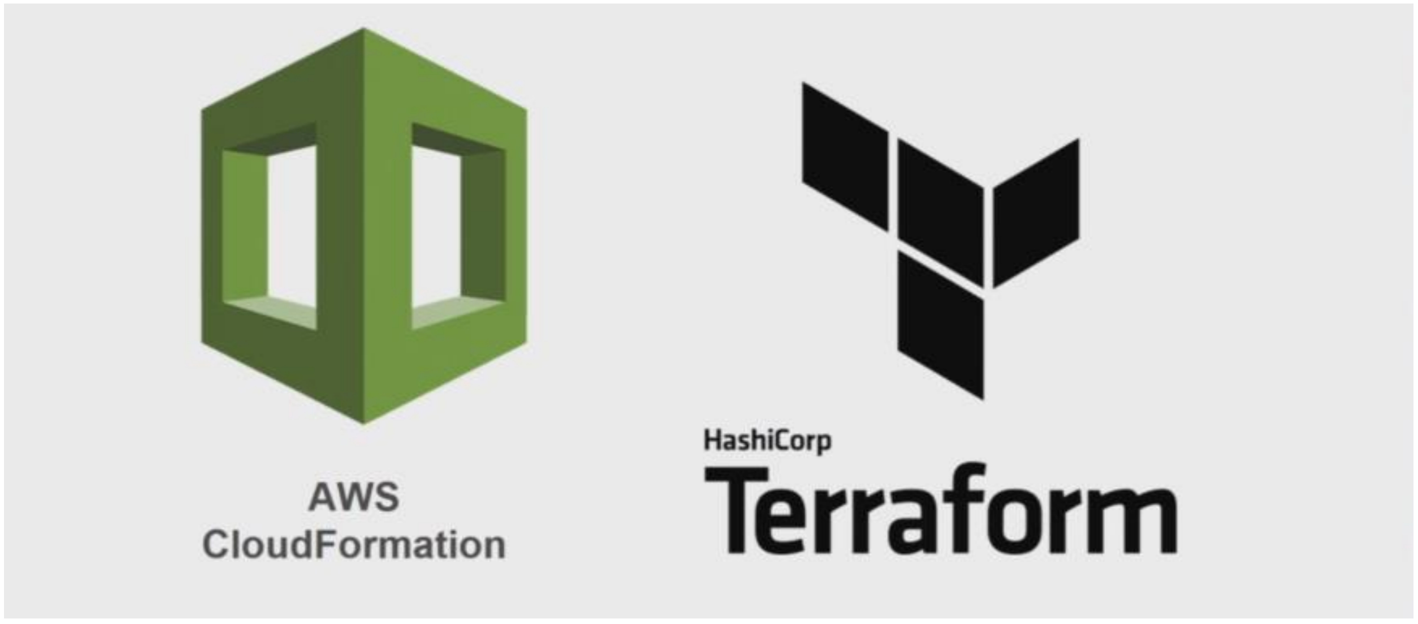
This leads to several services being built in the AWS platform that help build microservices



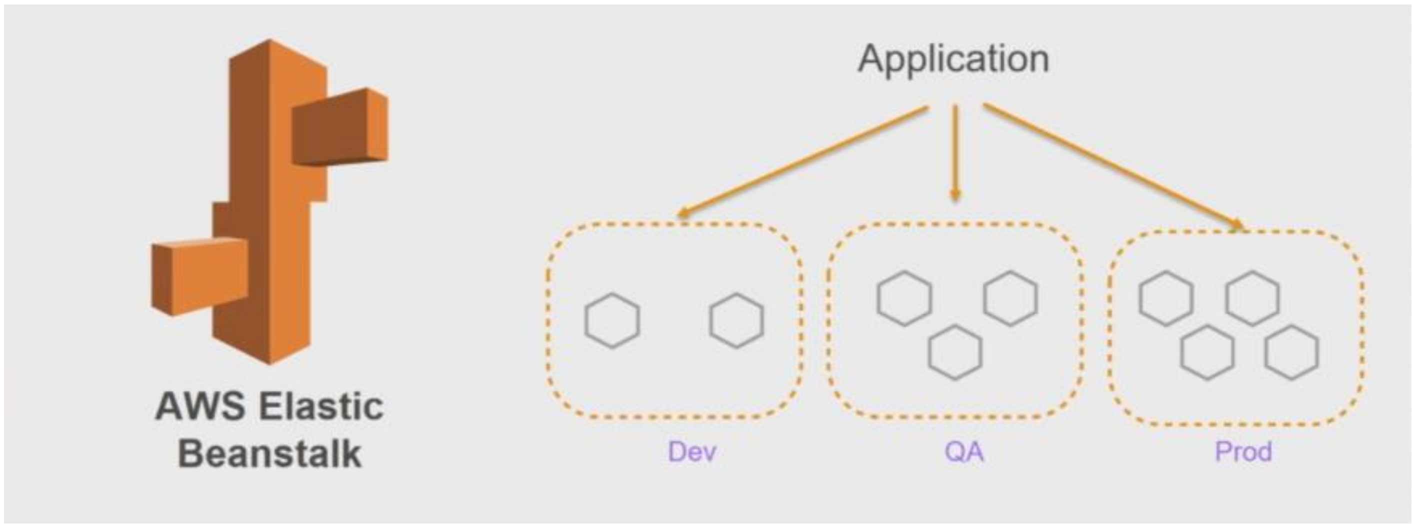
This is a very simple horizontally scaled microservice running in AWS that we have automated using a script


© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

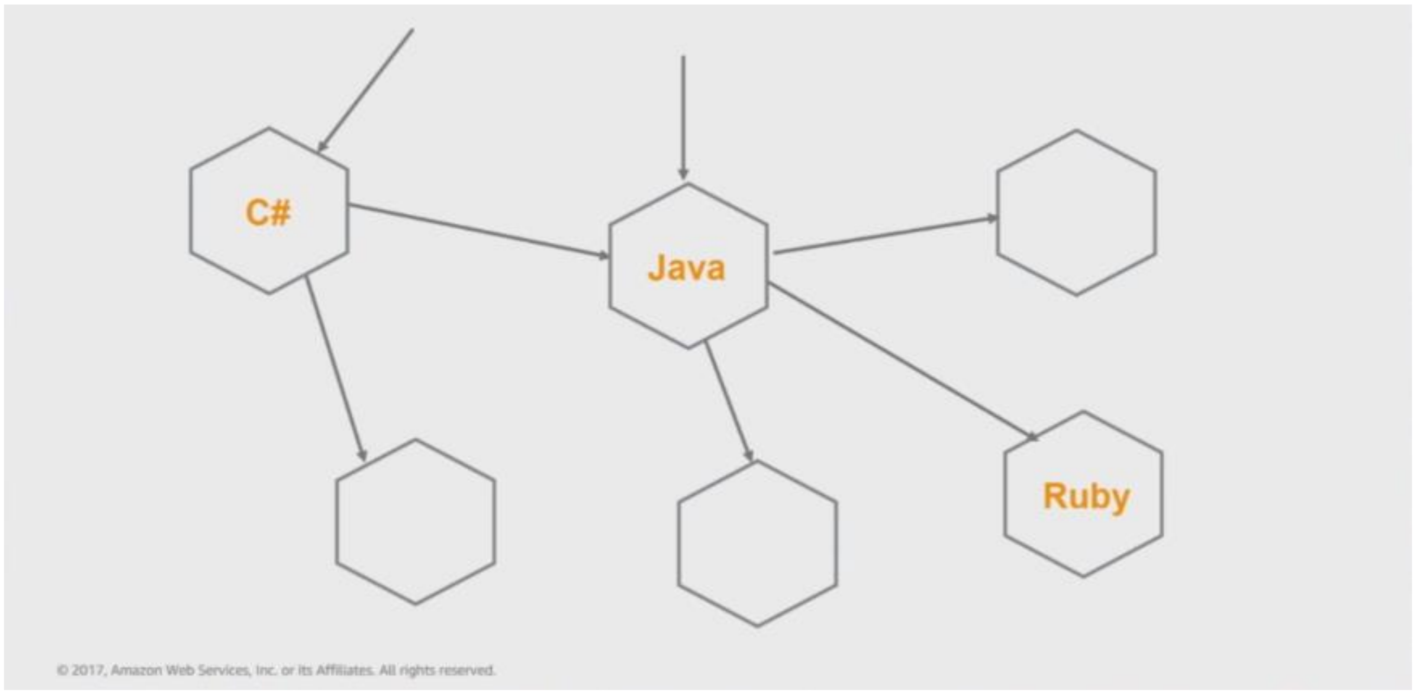
We can then have multiple microservices with different versions running for different uses in our platform, we start reaching for a higher order abstraction that helps us manage all these microservices



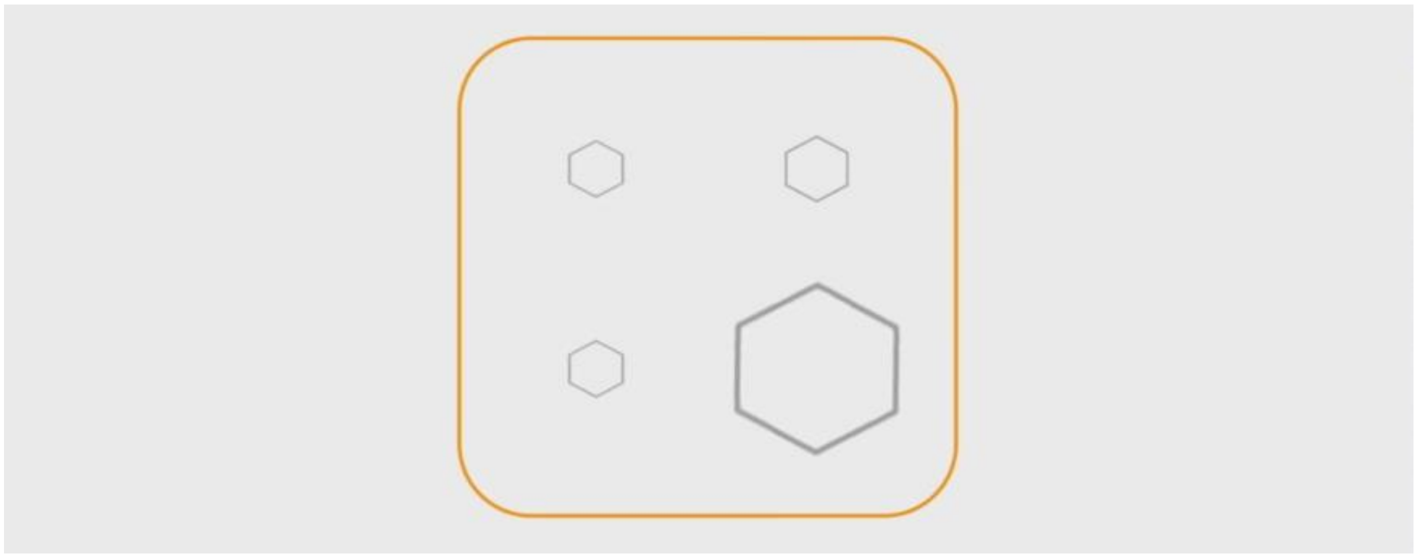
These allows us to create templates to underline our platform and allow us to create our services quickly and reliably



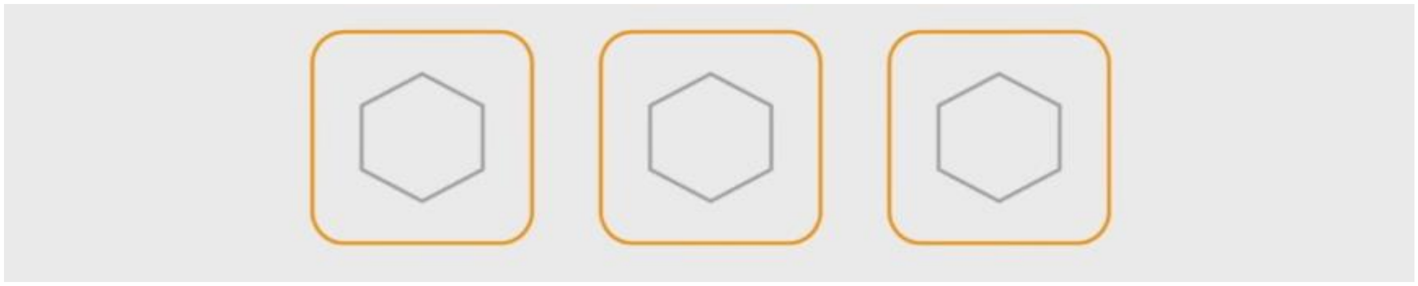
We can use EBS to run our application in any way we want, more for single monolithic applications than for services

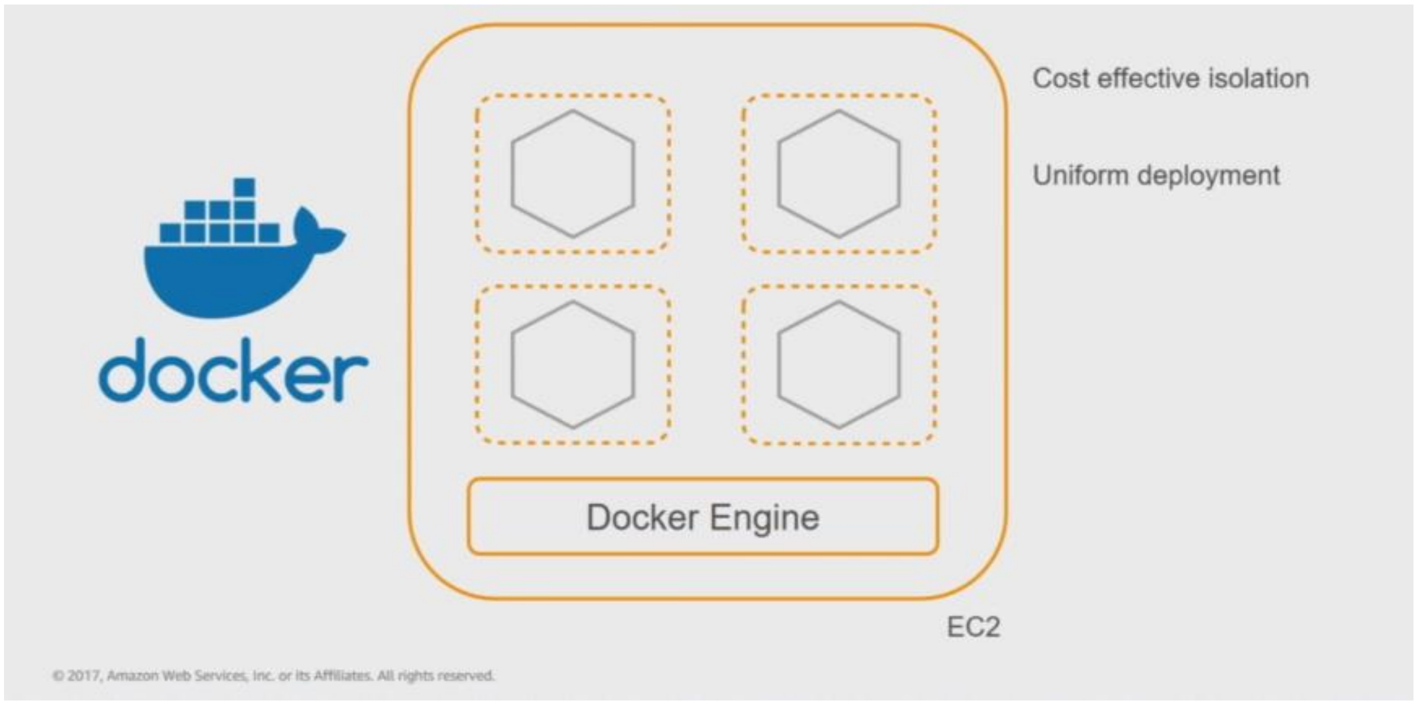

C#
Java
Ruby
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Containers can help by providing a uniform format for deploying all our apps by giving us the same unit of deployment

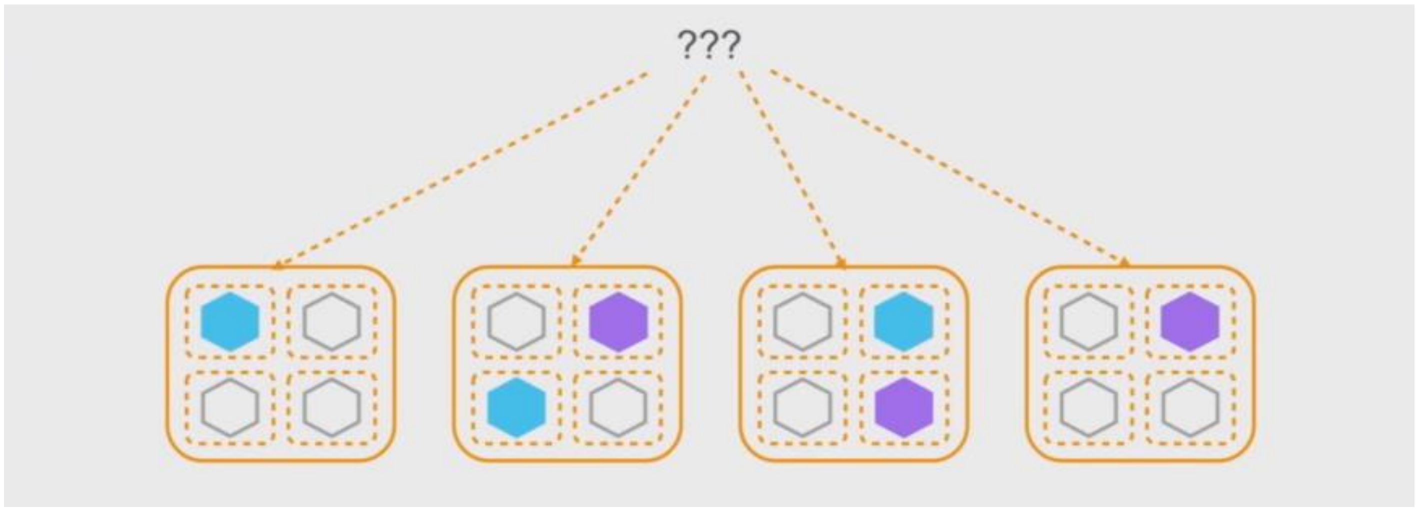


Services deployed together can have a single service getting more memory and resources and choking the other services, Docker containers allow us to deploy each service in its own isolated deployment unit or box.

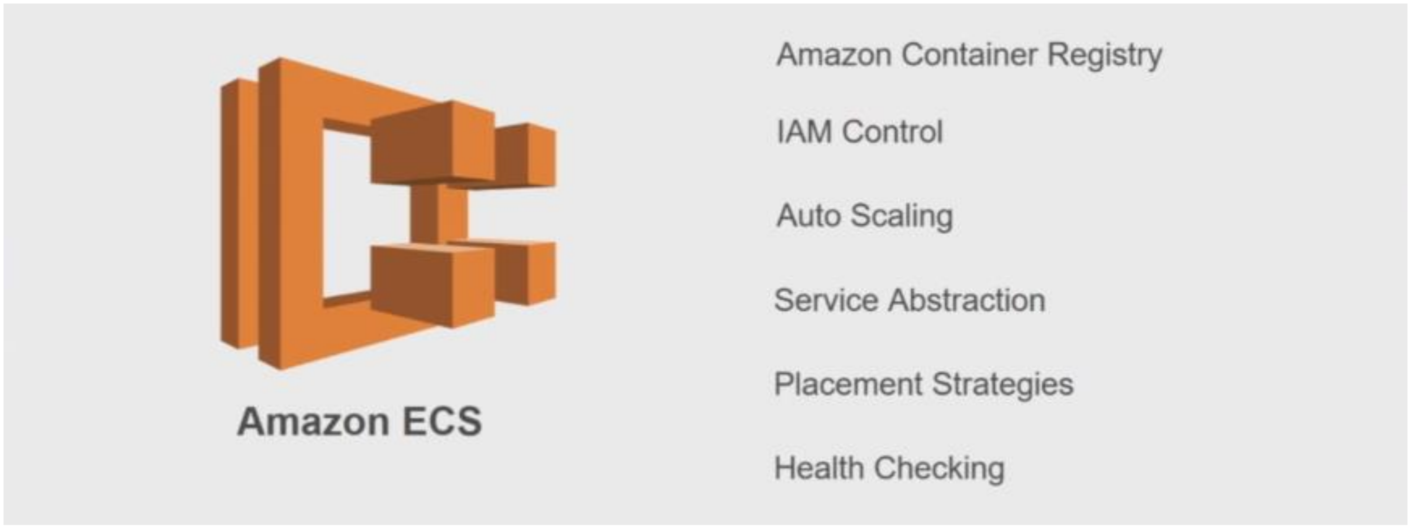



Cost effective isolation
Uniform deployment
docker
Docker Engine
EC2
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

We can launch a large EC2 instance for more I/O, then run the Docker engine locally inside the EC2 instance to use for deploying our other services as Docker containers. The problem here is that this is a single node solution that is not HA.



What we need is a way to launch multiple EC2 instances and then do the same deployment approach in them as above, this leads to the problem of managing this set of EC2 instances and the containers. This is a container scheduling problem.



Amazon ECS

- Amazon Container Registry
- IAM Control
- Auto Scaling
- Service Abstraction
- Placement Strategies
- Health Checking

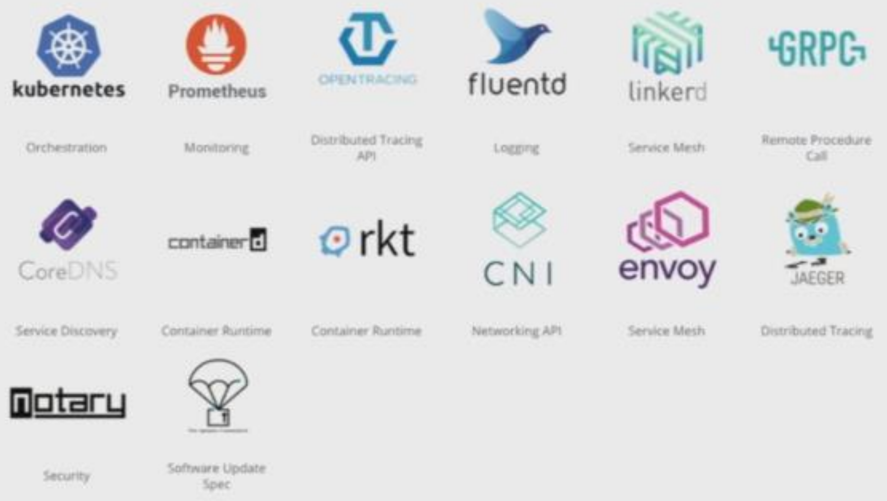
This is a way of managing multiple EC2 instances running Docker containers, this is a higher-level abstraction to allow us deploy applications over multiple EC2 instances. ECS also gives you all the benefits listed above like service abstraction using EBS, auto scaling, IAM control, placement strategies like bean-packing or spread etc.

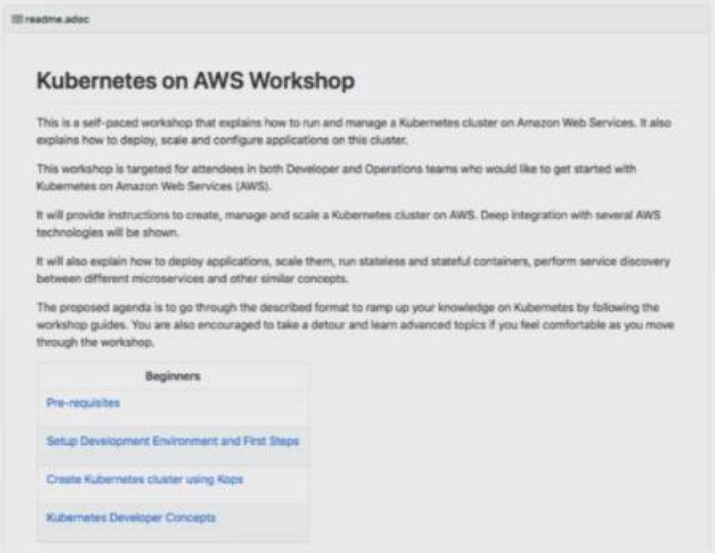
Kubernetes gives you portable across multiple providers and on-premises

CLOUD NATIVE
COMPUTING FOUNDATION

CNCF is a body trying to make it easy for people to build cloud native applications efficiently

CNCF projects





readme.adoc

Kubernetes on AWS Workshop

This is a self-paced workshop that explains how to run and manage a Kubernetes cluster on Amazon Web Services. It also explains how to deploy, scale and configure applications on this cluster.

This workshop is targeted for attendees in both Developer and Operations teams who would like to get started with Kubernetes on Amazon Web Services (AWS).

It will provide instructions to create, manage and scale a Kubernetes cluster on AWS. Deep integration with several AWS technologies will be shown.

It will also explain how to deploy applications, scale them, run stateless and stateful containers, perform service discovery between different microservices and other similar concepts.

The proposed agenda is to go through the described format to ramp up your knowledge on Kubernetes by following the workshop guides. You are also encouraged to take a detour and learn advanced topics if you feel comfortable as you move through the workshop.

Beginners
Pre-requisites
Setup Development Environment and First Steps
Create Kubernetes cluster using Kops
Kubernetes Developer Concepts

https://github.com/arun-gupta/kubernetes-aws-workshop

AWS re:Invent

aws

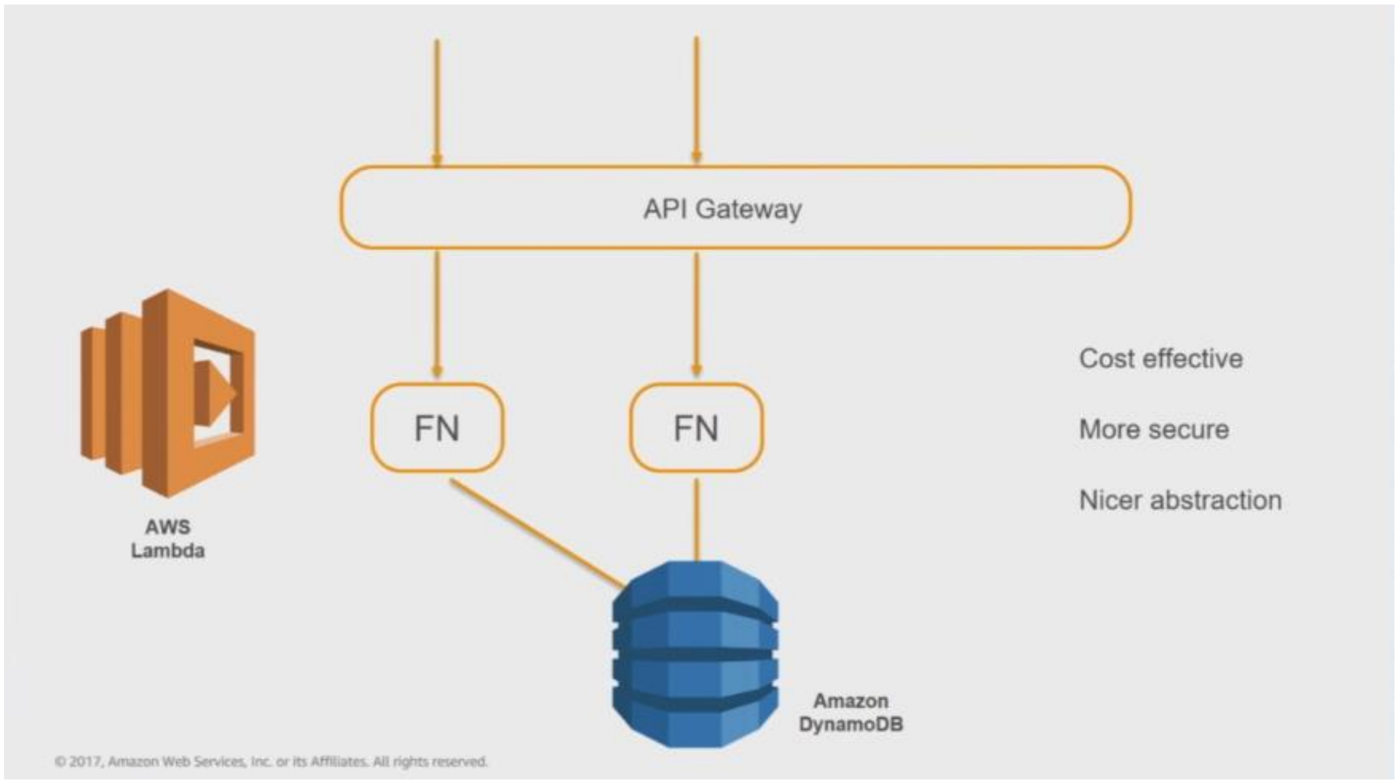
#serverless

“The phrase 'serverless' doesn’t mean servers are no longer involved. It simply means that developers no longer have to think that much about them.”

—Ken Fromm

http://readwrite.com/2012/10/15/why-the-future-of-software-and-apps-is-serverless/

AWS Lambda **Amazon DynamoDB** **Amazon S3**



#serverless

Better, developer-friendly abstractions

Functions as a service =
Containers as implementation detail

aws *+ Microservices*

http://samnewman.io/

@samnewman

Thanks!