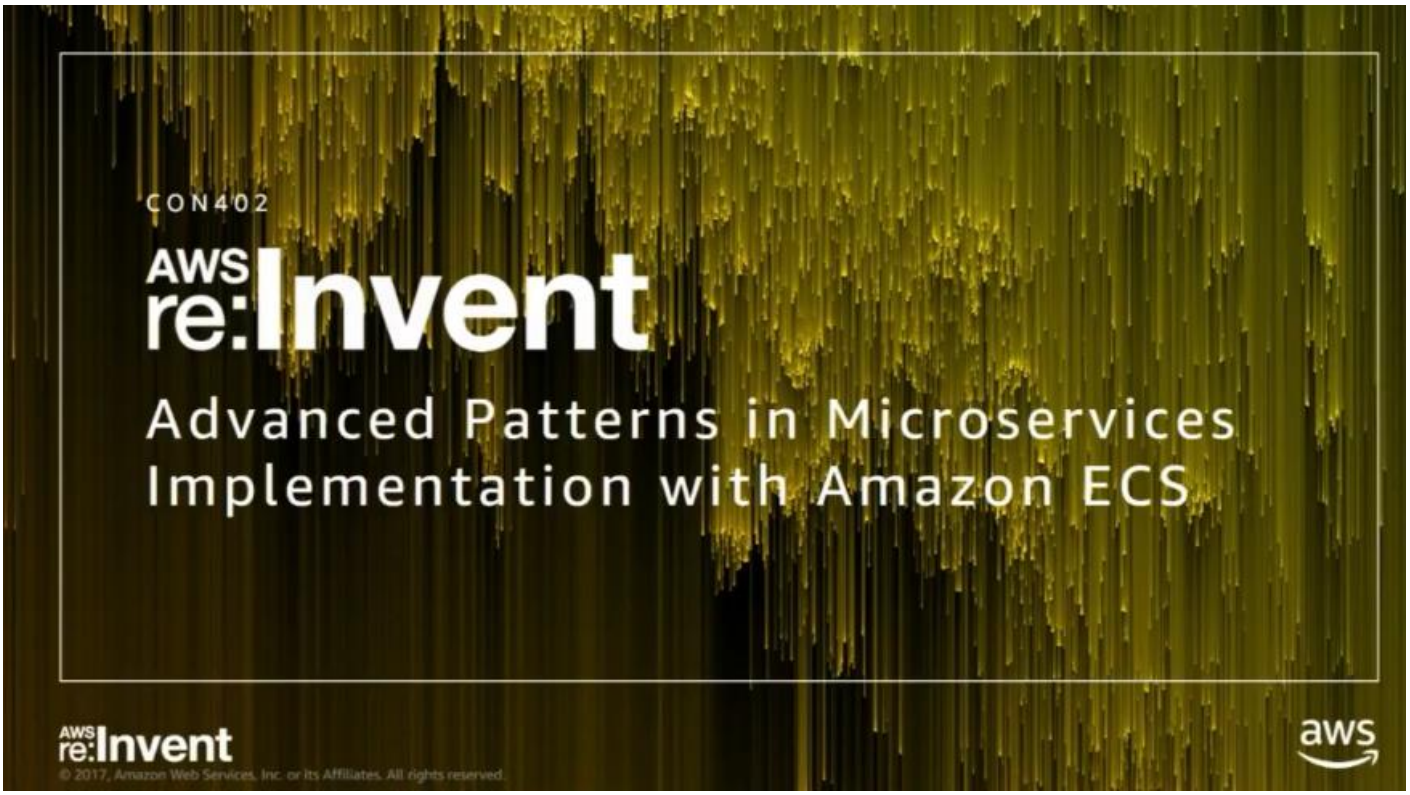
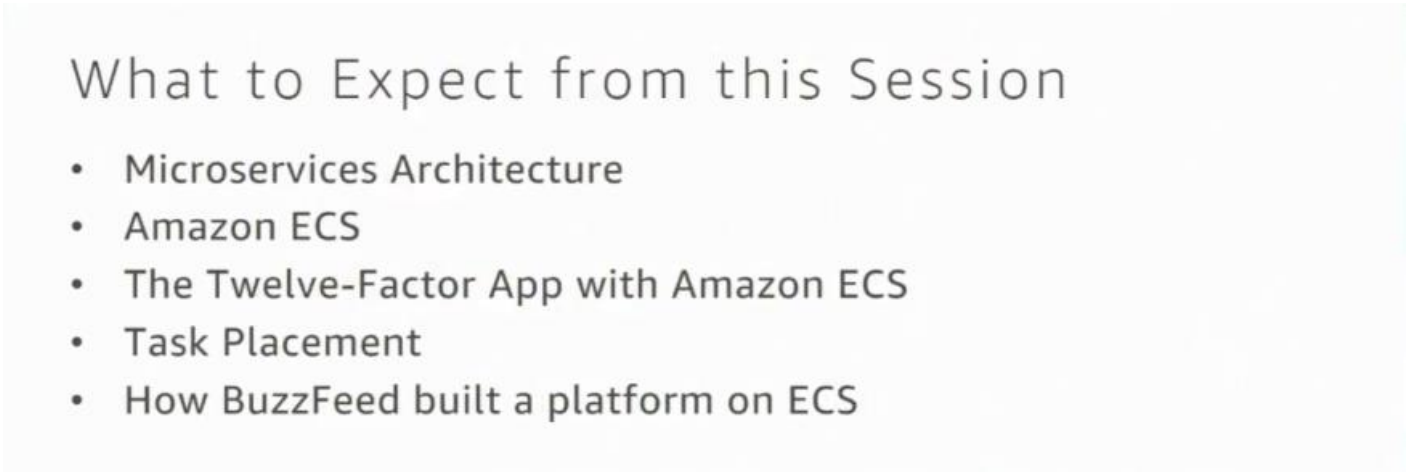


Scaling a microservice-based infrastructure can be challenging in terms of both technical implementation and developer workflow. In this talk, AWS Solutions Architect Pierre Steckmeyer will be joined by Will McCutchen, Architect at BuzzFeed, to discuss Amazon ECS as a platform for building a robust infrastructure for microservices. We will look at the key attributes of microservice architectures and how Amazon ECS supports these requirements in production, from configuration to sophisticated workload scheduling to networking capabilities to resource optimization. We will also examine what it takes to build an end-to-end platform on top of the wider AWS ecosystem, and what it's like to migrate a large engineering organization from a monolithic approach to microservices.



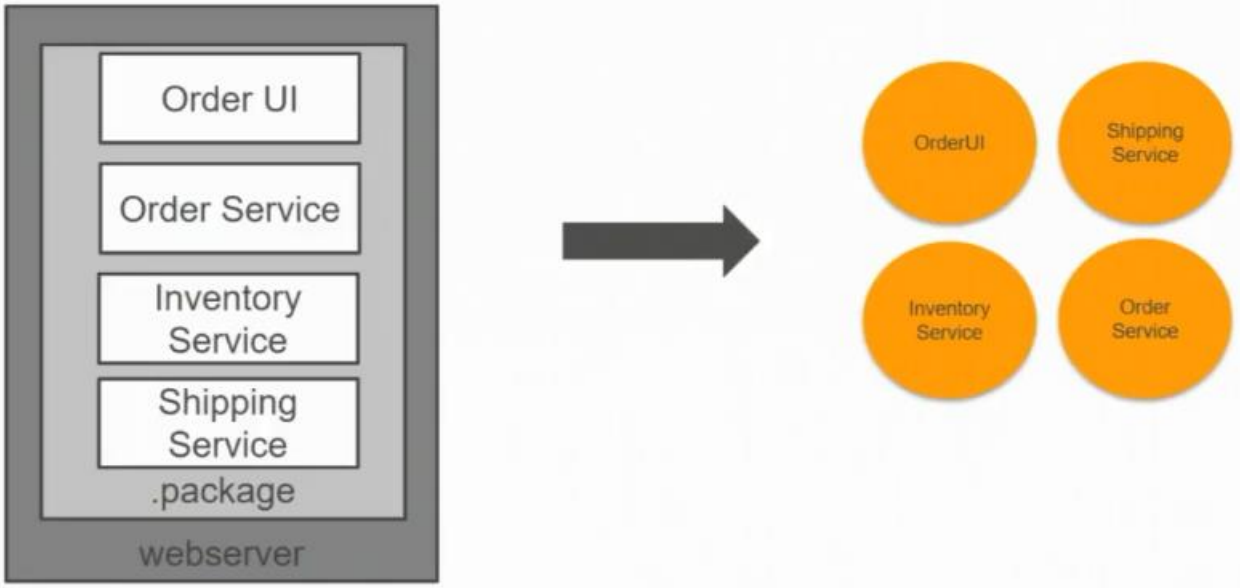
What are Microservices?

Microservices are an **architectural and organizational** approach to software development in which software is composed of **small, independent services** that communicate over **well-defined APIs**. These services are owned by **small, self-contained** teams.

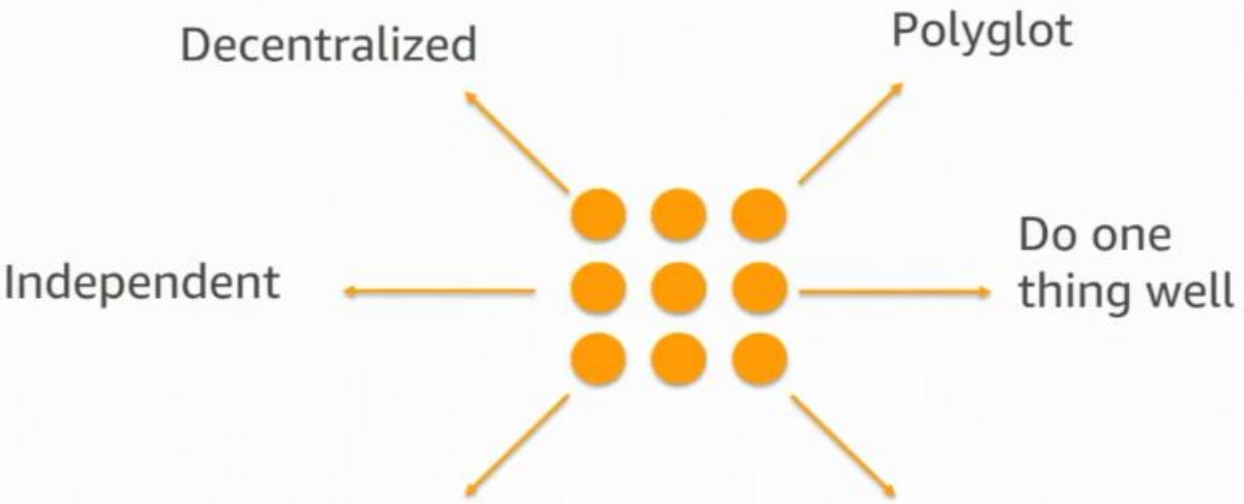
Watch the replay: CON208

Whitepaper: http://bit.ly/2A0qGdt - Running Containerized Microservices on AWS

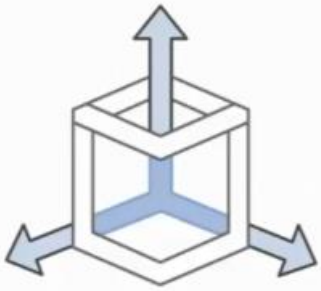
Monolithic vs. Microservices



Characteristics



Why Amazon ECS



- Fully managed elastic service – You don't need to run anything, and the service scales as your microservices architecture grows
- Shared state optimistic scheduling
- Integration with Amazon CloudWatch service for monitoring and logging
- Integration with Code* services for continuous integration and delivery (CI/CD)

Deploying Containers on ECS – Choose a Scheduler

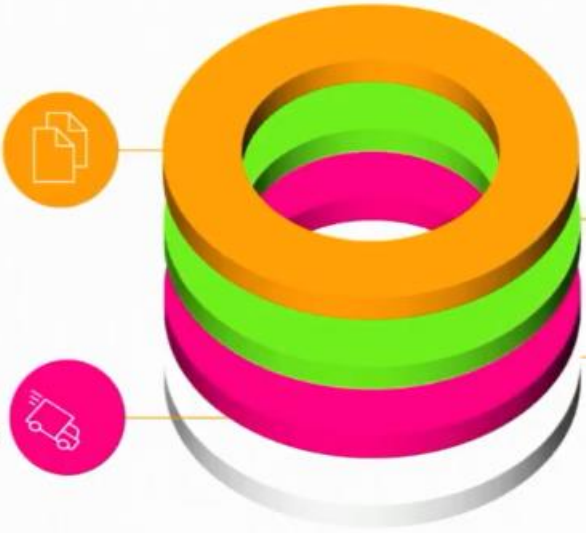
Batch Jobs	Long-Running Apps
ECS task scheduler	ECS service scheduler
Run tasks once	Health management
Batch jobs	Scale-up and scale-down
RunTask (random)	AZ aware
StartTask (placed)	Grouped containers

Reference Architectures

The Twelve-Factor App with Amazon ECS

The Twelve-Factor App

1. Codebase
2. Dependencies
3. Config

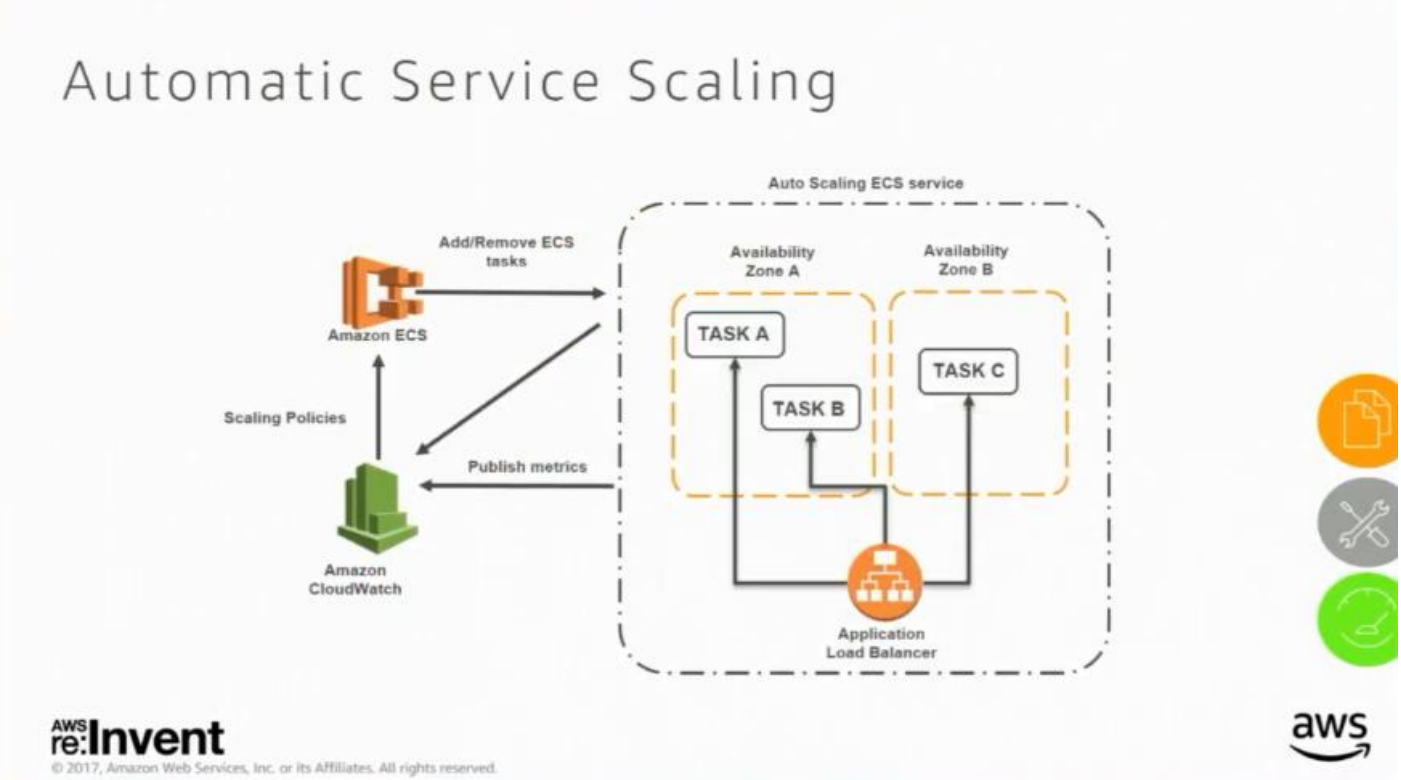


7. Port Binding
8. Concurrency
9. Disposability

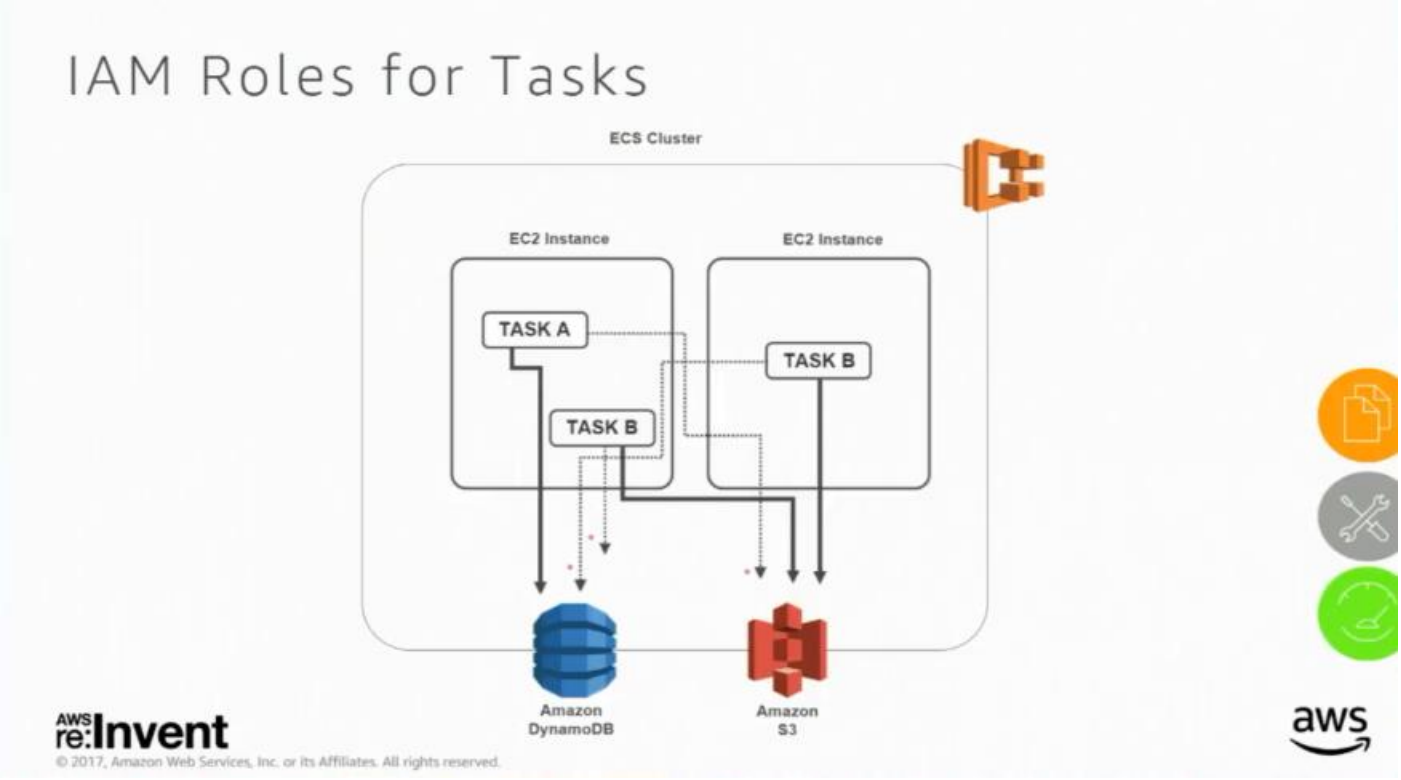
10. Dev/Prod parit
11. Logs
12. Admin Process

4. Backing Services
5. Build, Release Run
6. Processes

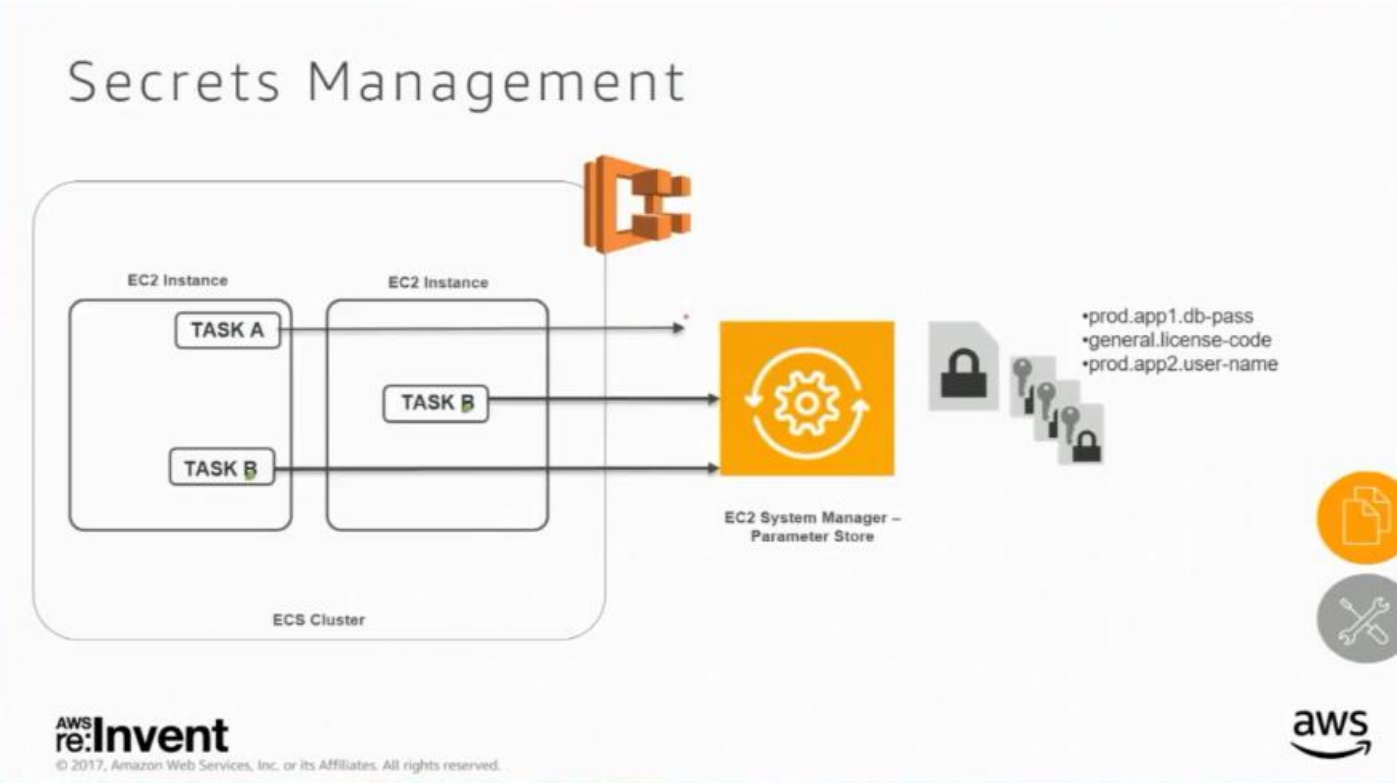
https://12factor.net/



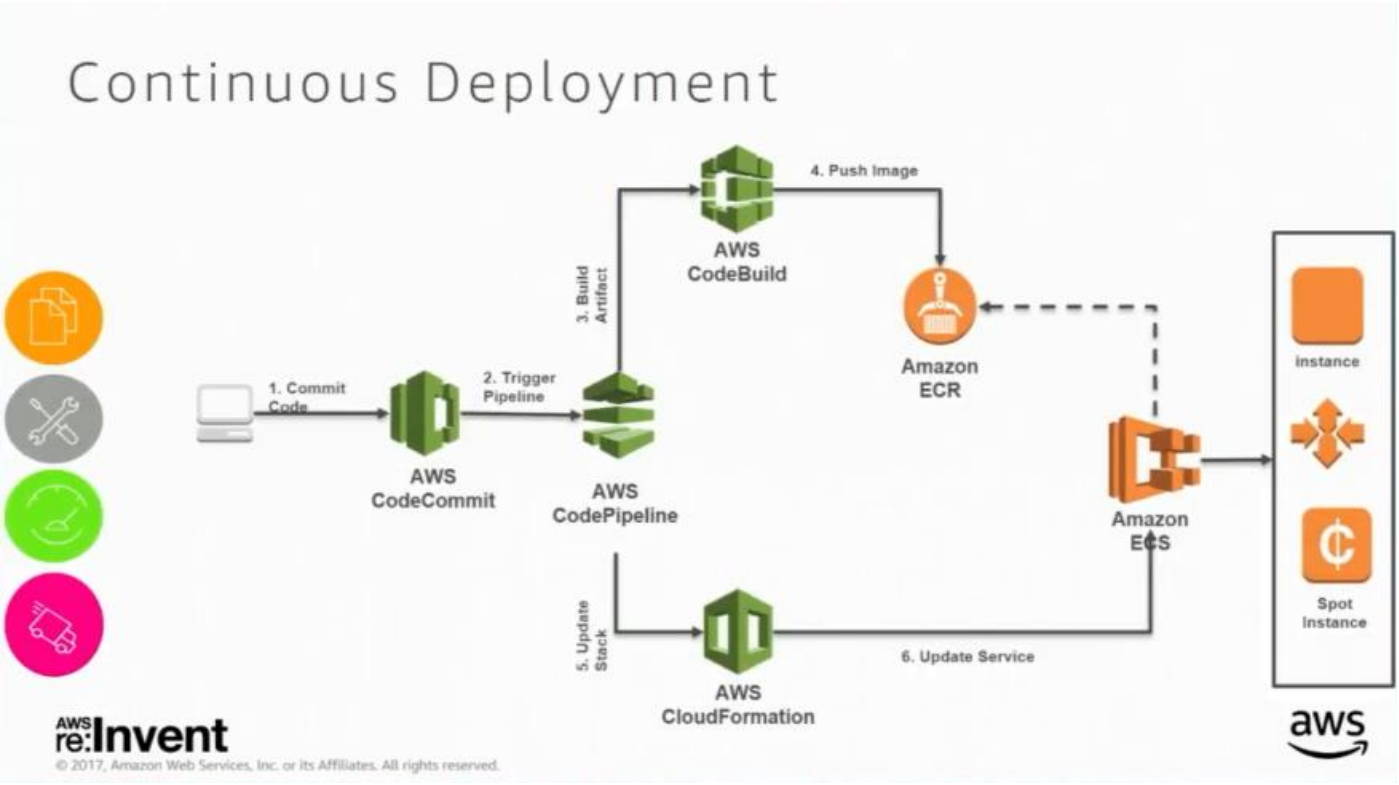
ECS recommends that you scale across 2 dimensions. The 1st dimension is at the app level or how many tasks are you running? And at the cluster level, how many instances are you running within it.



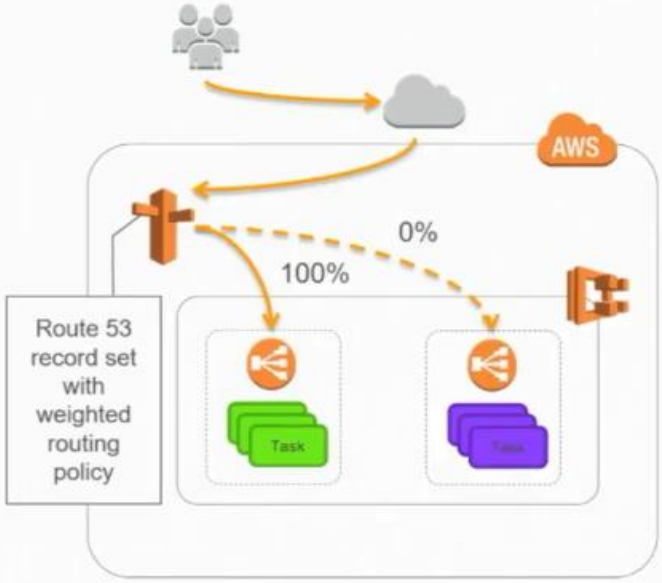
You can now give IAM roles to your tasks, this means that you can have a task that is allowed to read from S3 only and another task that is allowed to read and write to DynamoDB.



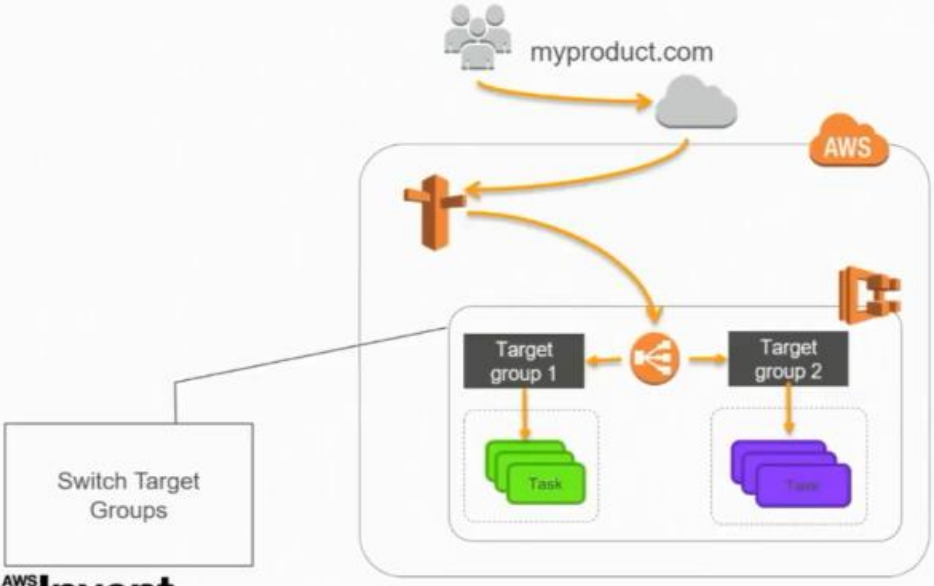
You can encrypt the password you pass to your task and give the task an IAM role that has the key to decrypt the password using the system manager

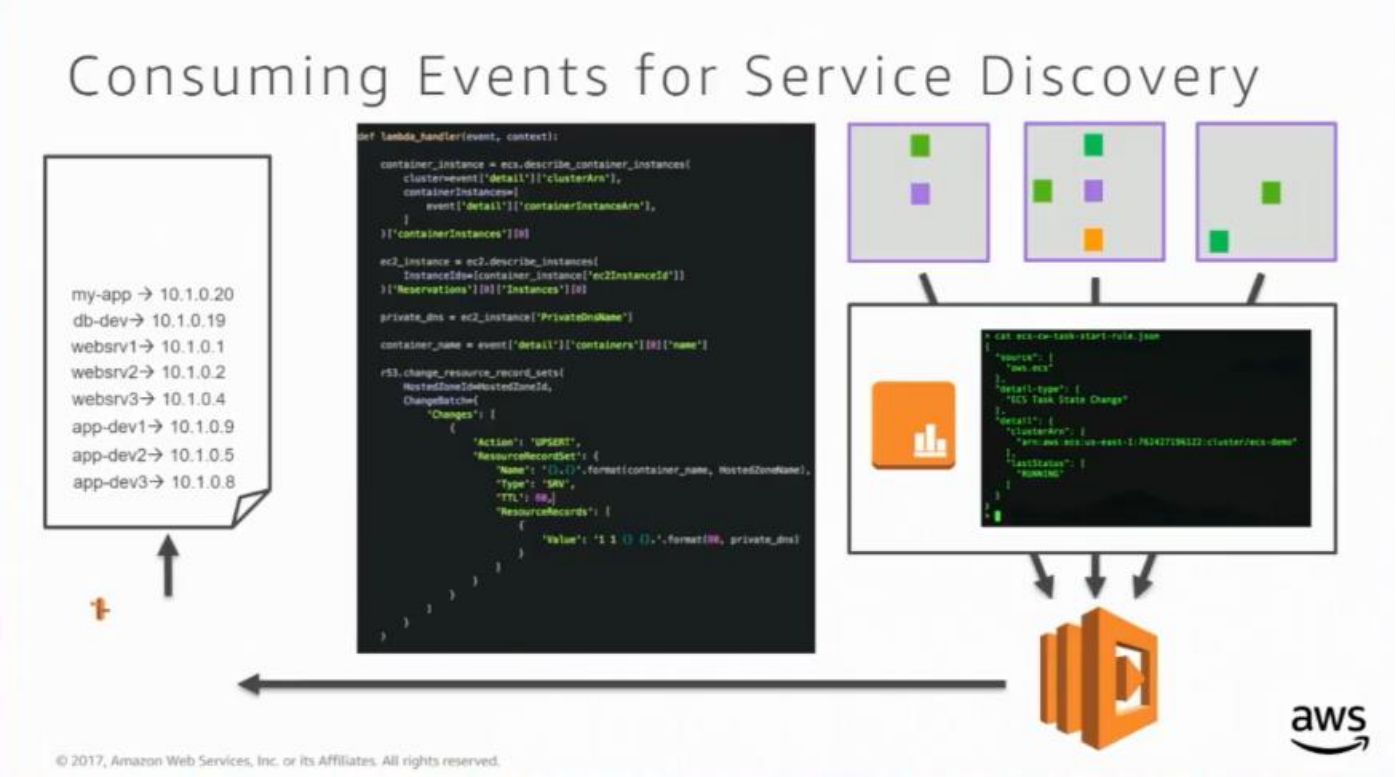


Blue-Green Deployments (DNS-based)

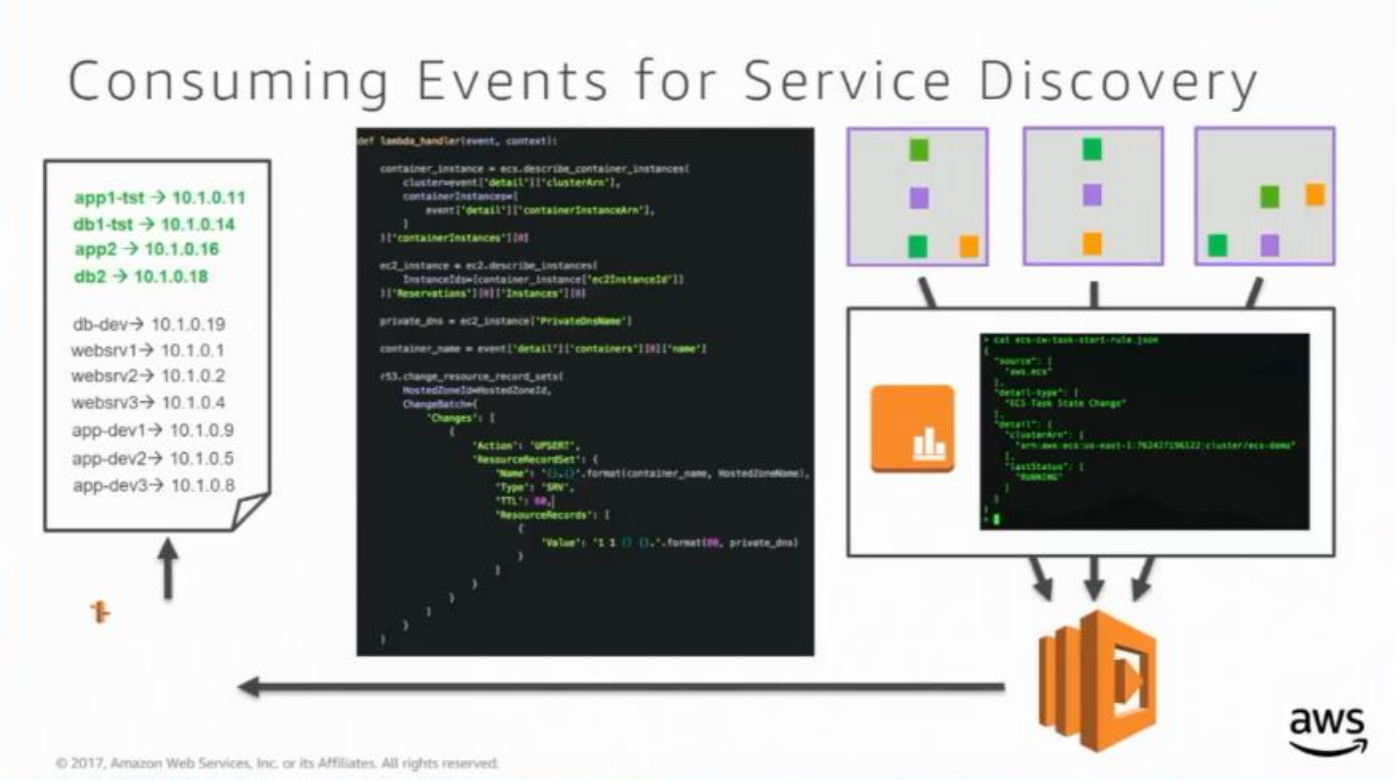


Blue-Green Deployments (Target Group Switch)

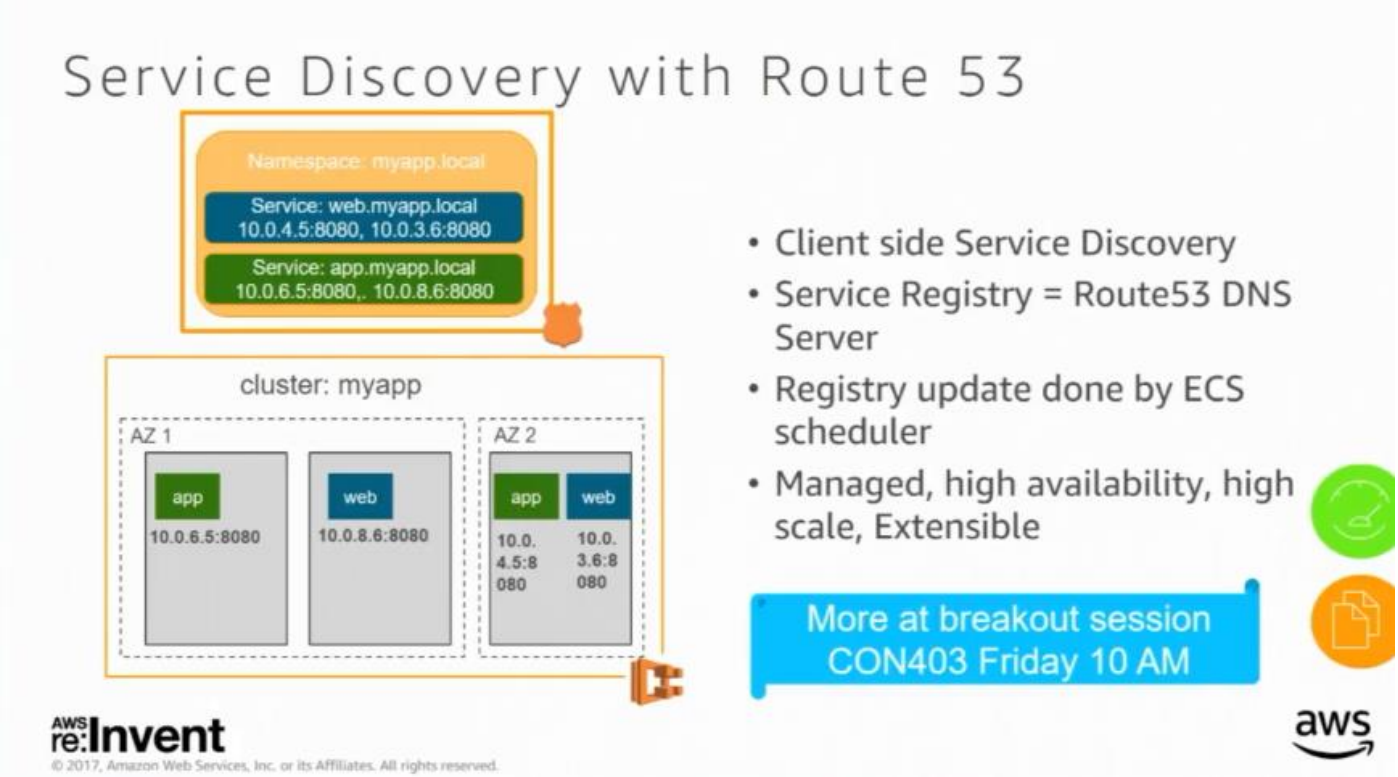




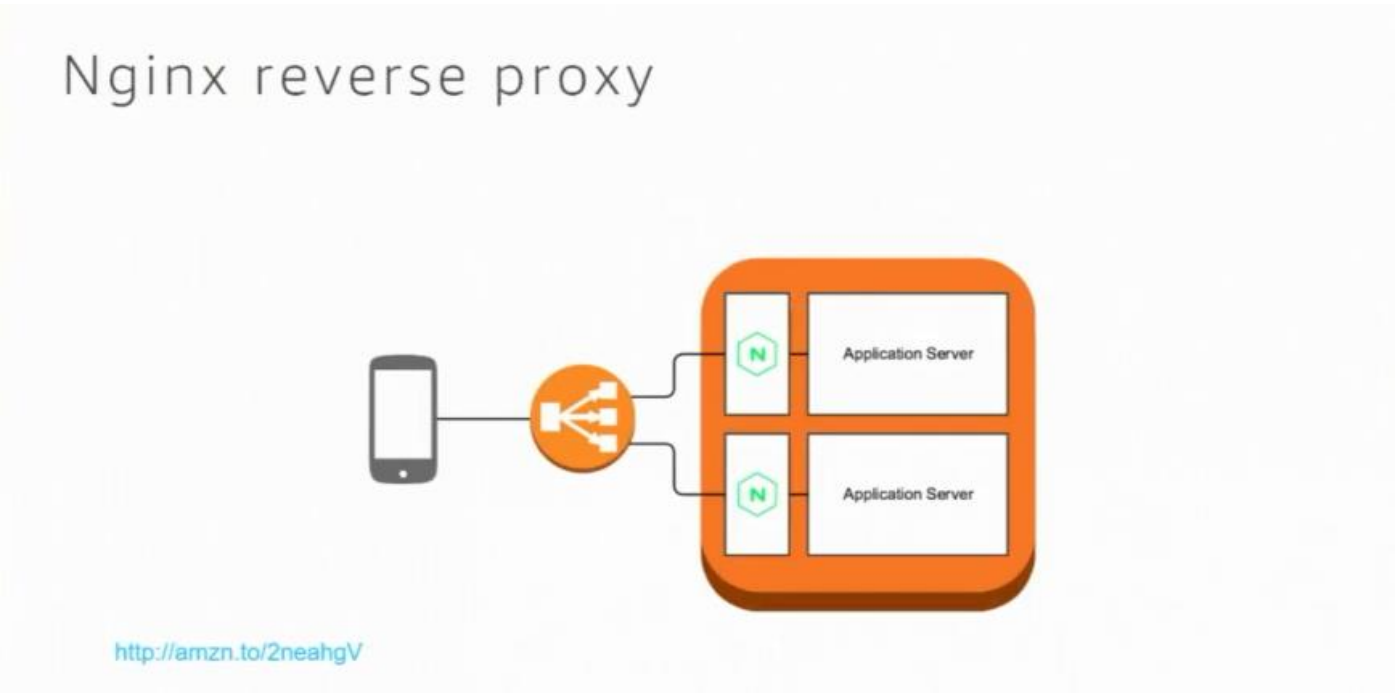
For a DIY way to do service discovery, we can use the event stream that comes out of ECS



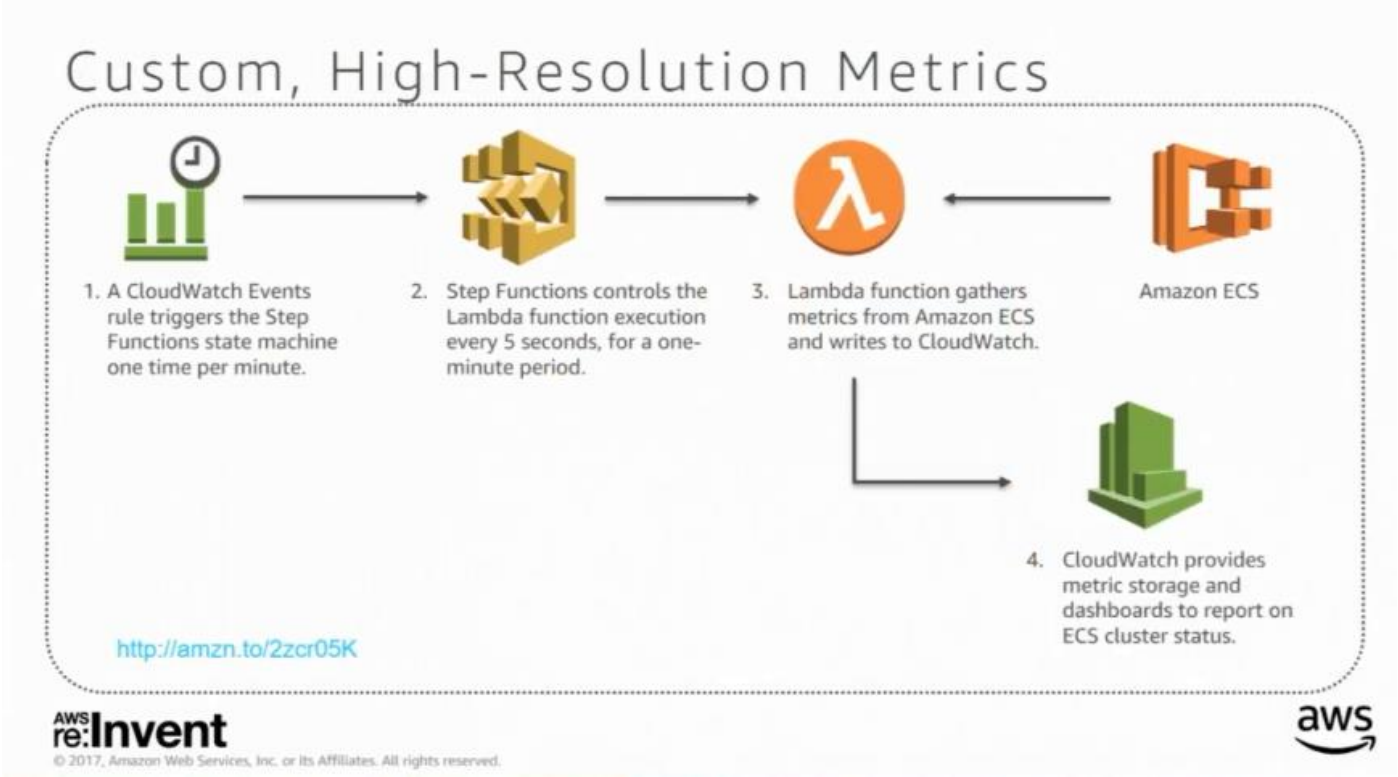
We can use lambda and route53 hosted zones as things happen

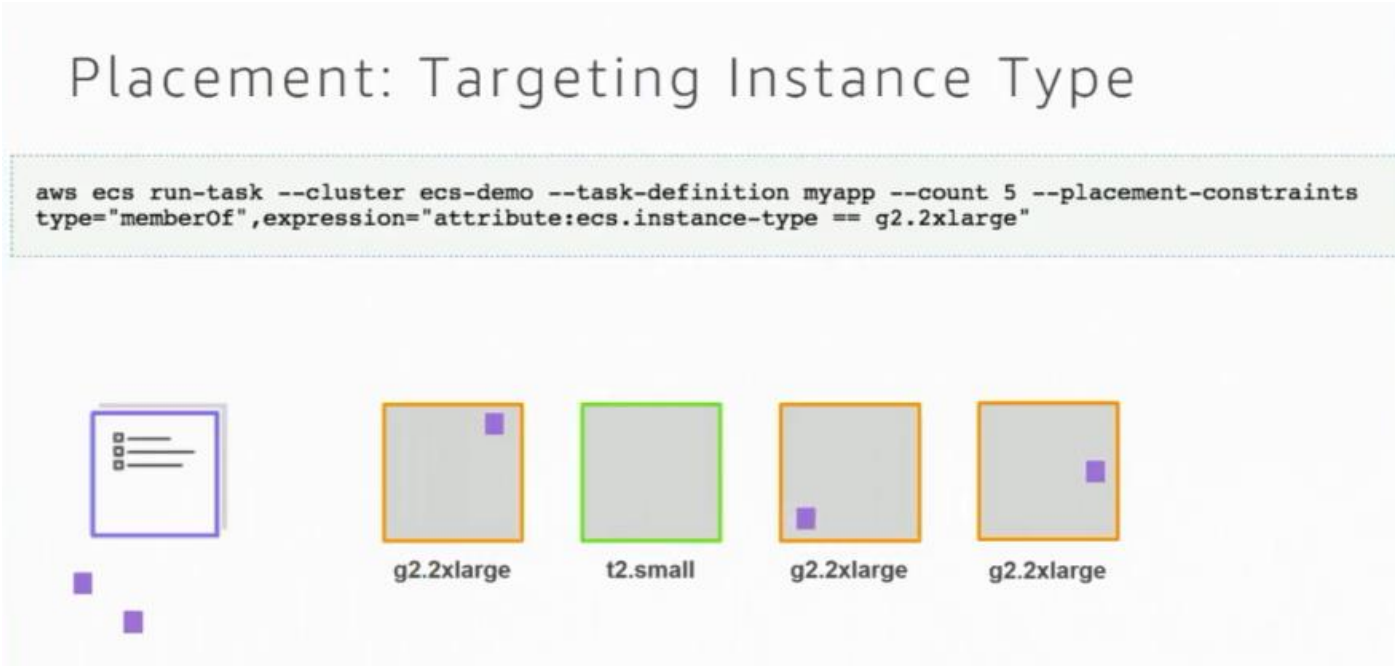


Route53 is now going to start supporting service discovery, you just have to tick a box when starting up ECS and Route53 will start taking note of the instances coming up and going

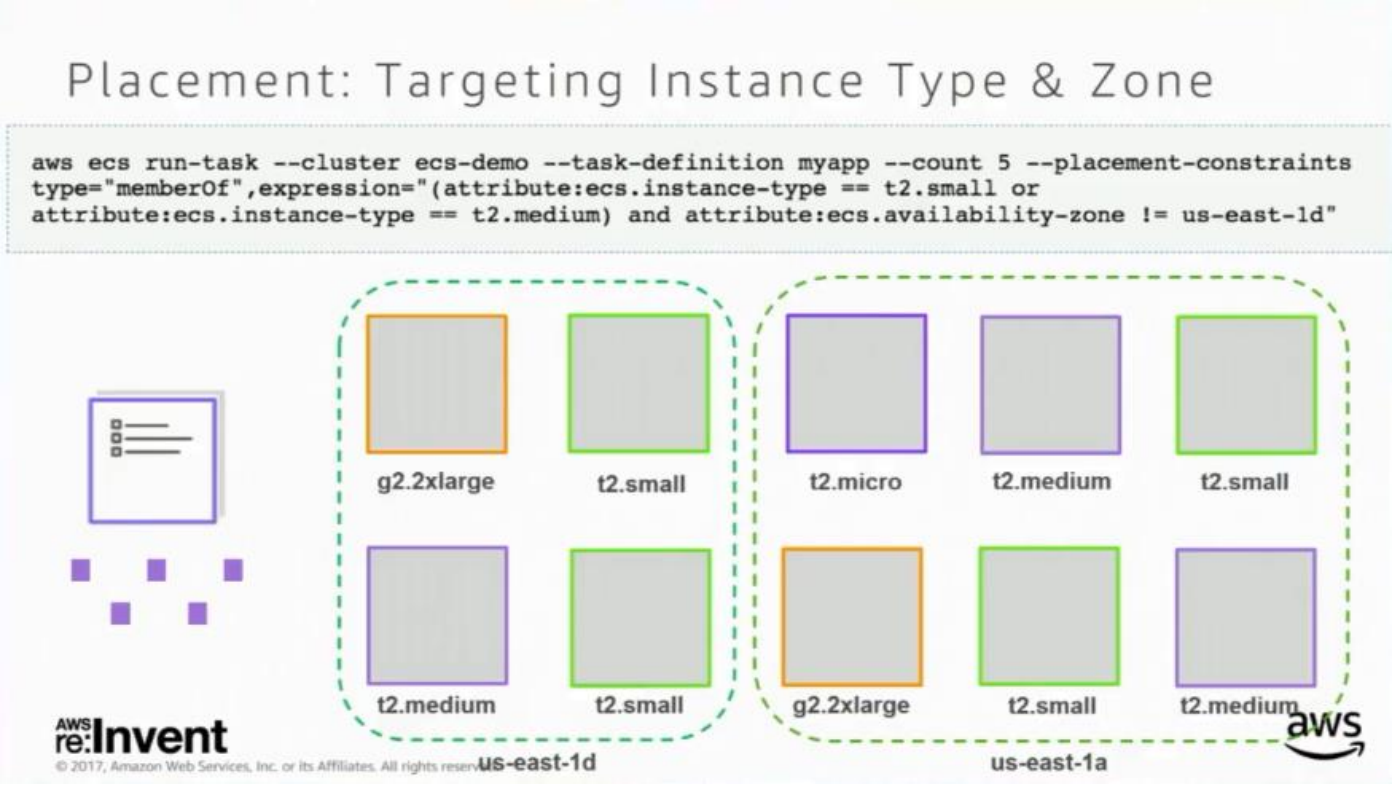


Nginx can provide security by filtering requests that you want to allow and reject all other requests, you can also use Nginx for gzip compression

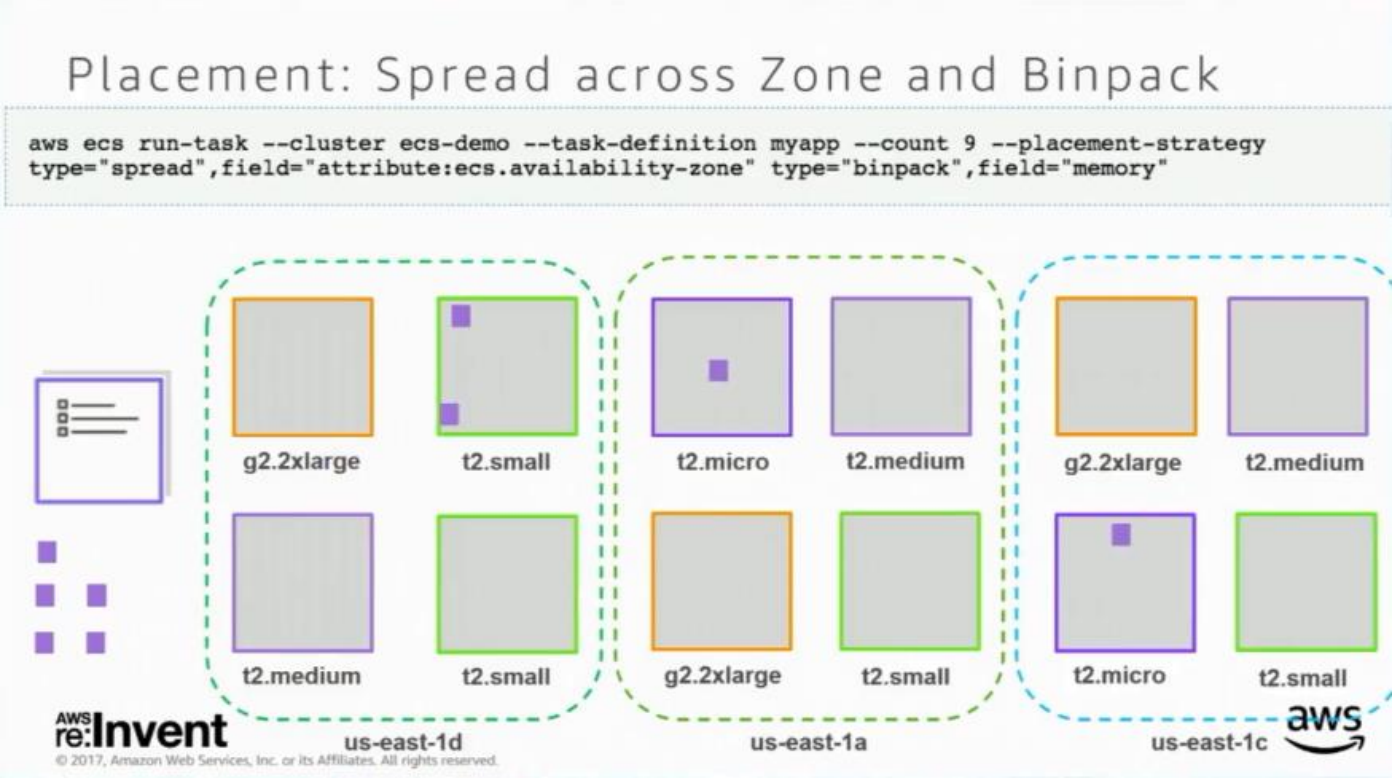




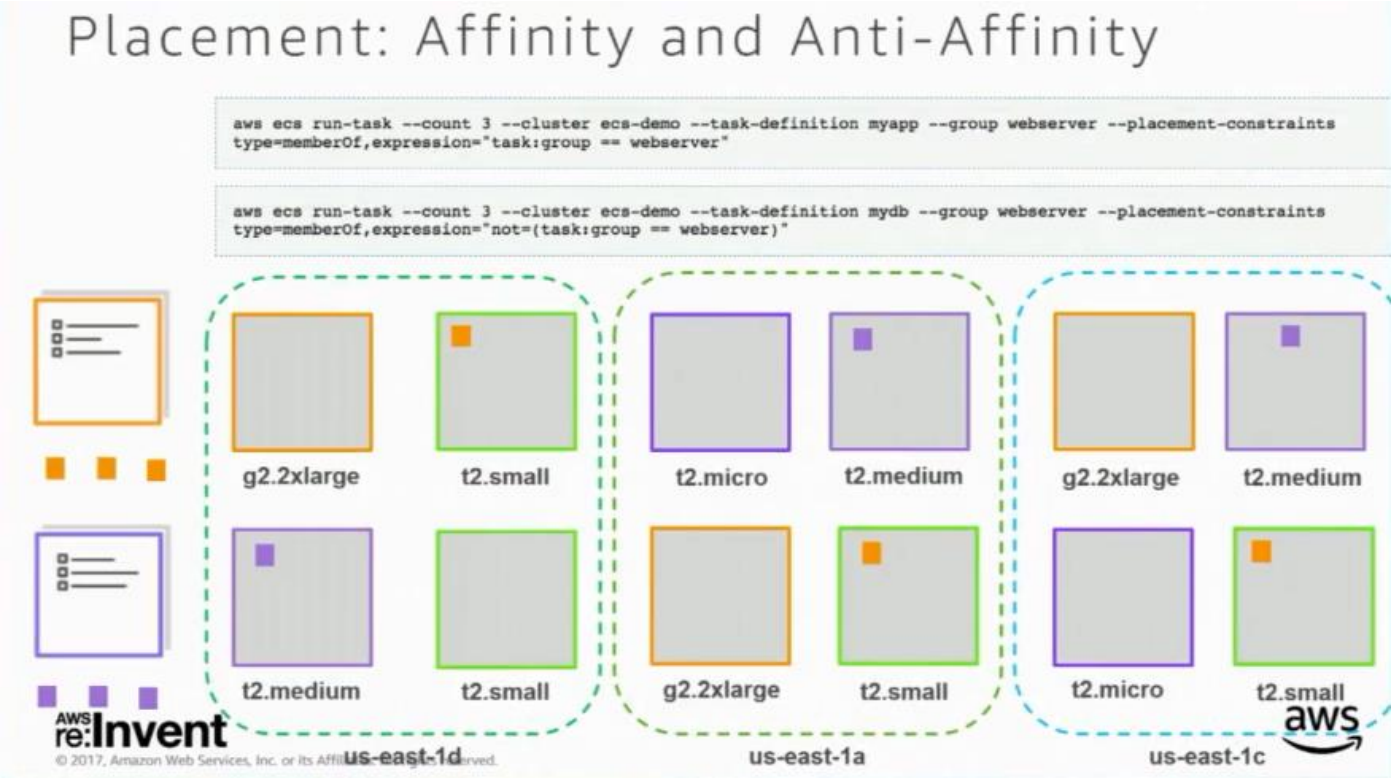
You can indicate that you want your task to land of some specific instance type



You can also specify you want your container to land on certain AZ



You can also spread your workload on different AZ and also do bin packing



You can also specify if you want tasks to be close to one another or not

You can do this at runtime or just embed it within your service definition file as above

BuzzFeed

Building a platform on ECS

BuzzFeed

Will McCutchen
Platform Infrastructure

will.mccutchen@buzzfeed.com
twitter.com/mccutchen

README.md

rig

> *v.* Set up (equipment or a device or structure), typically hastily or in makeshift fashion

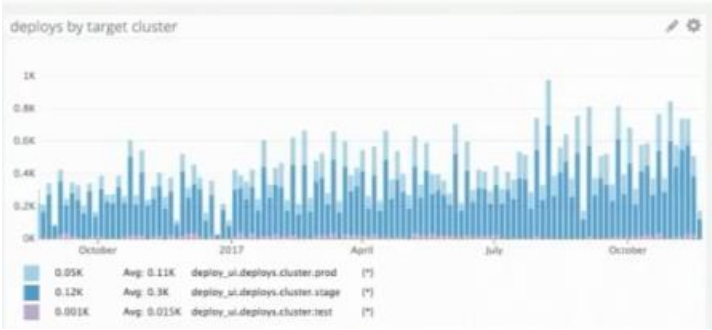
`rig` is a complete platform for containerized services on Amazon ECS.

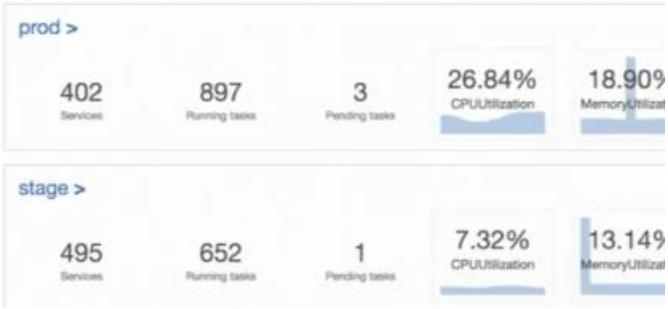
View all of README.md

Where are we now?

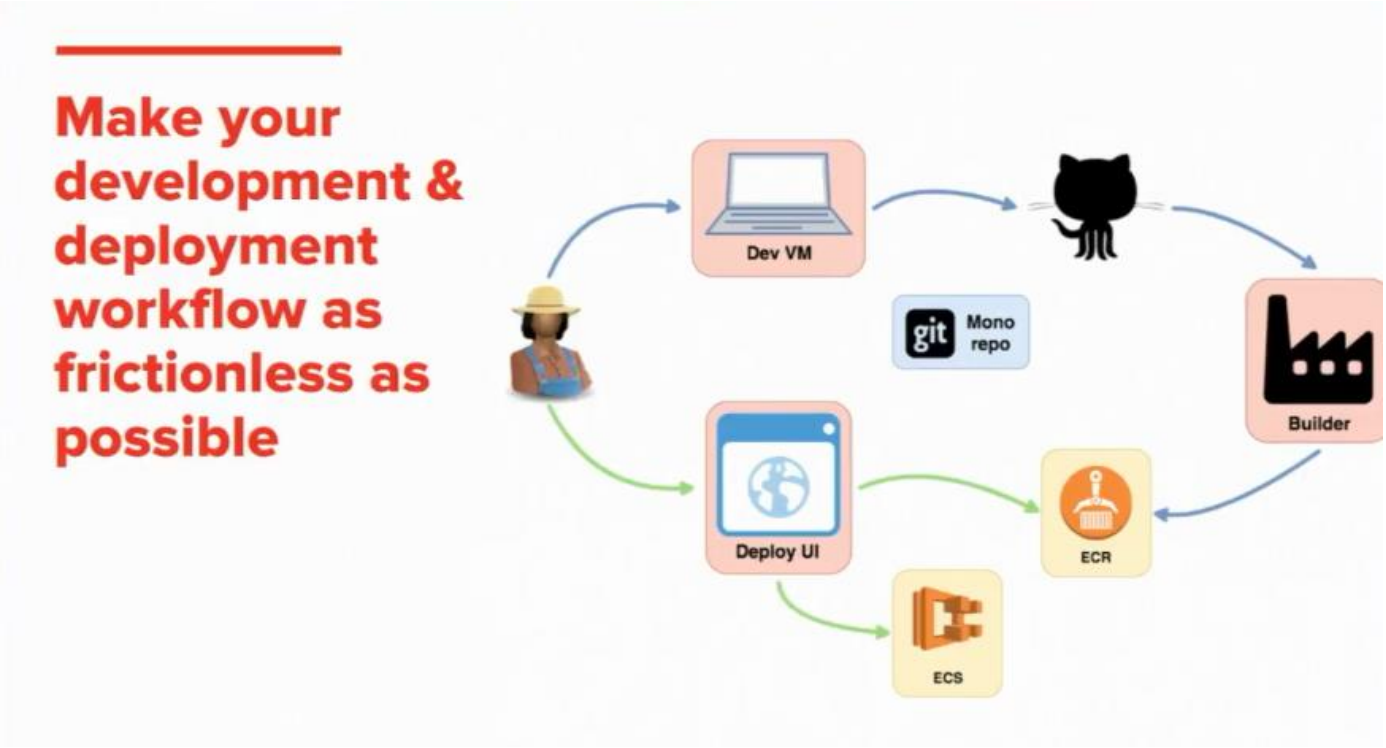
- ~20 months in production
- ~200 users
- > 50,000 deploys

- ~400 services currently deployed
- ~80 container instances
- 6 clusters
- 2 regions

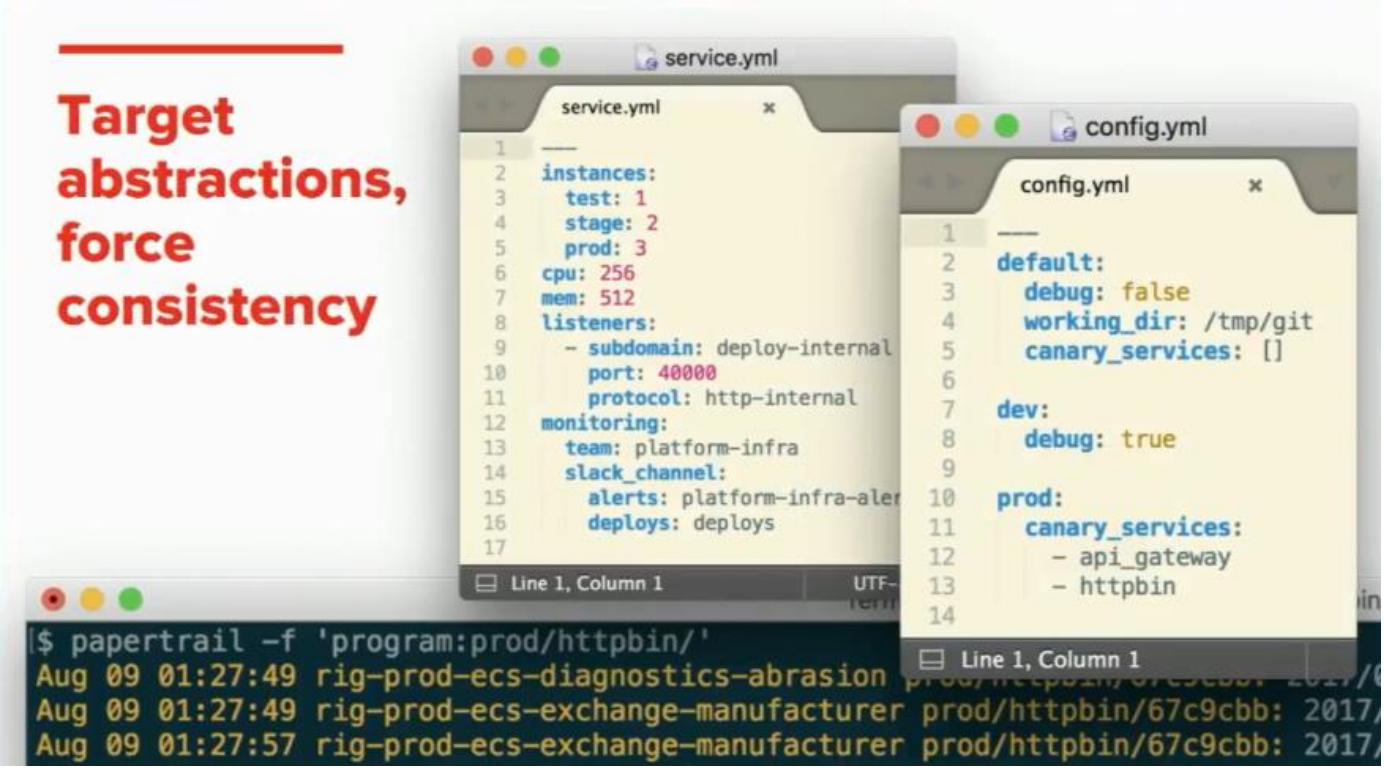




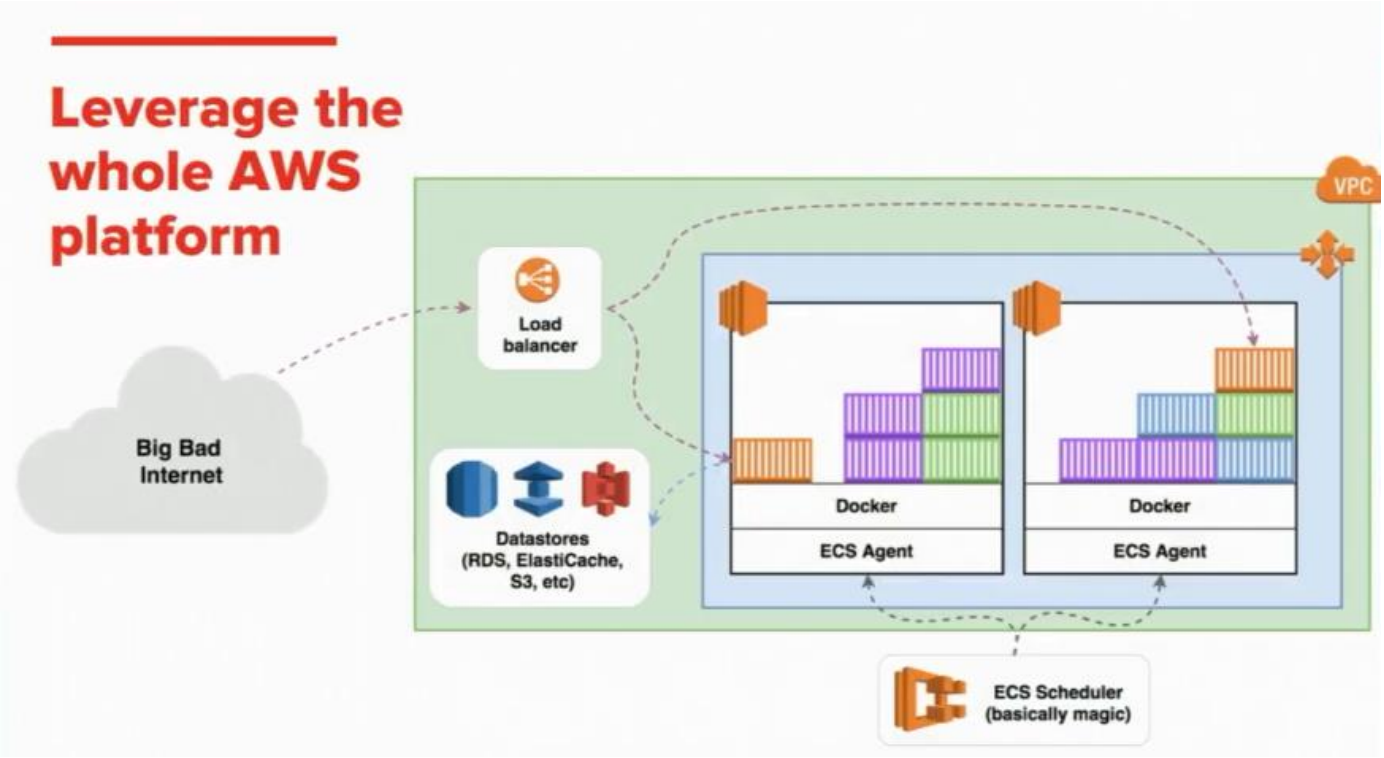
What we learned



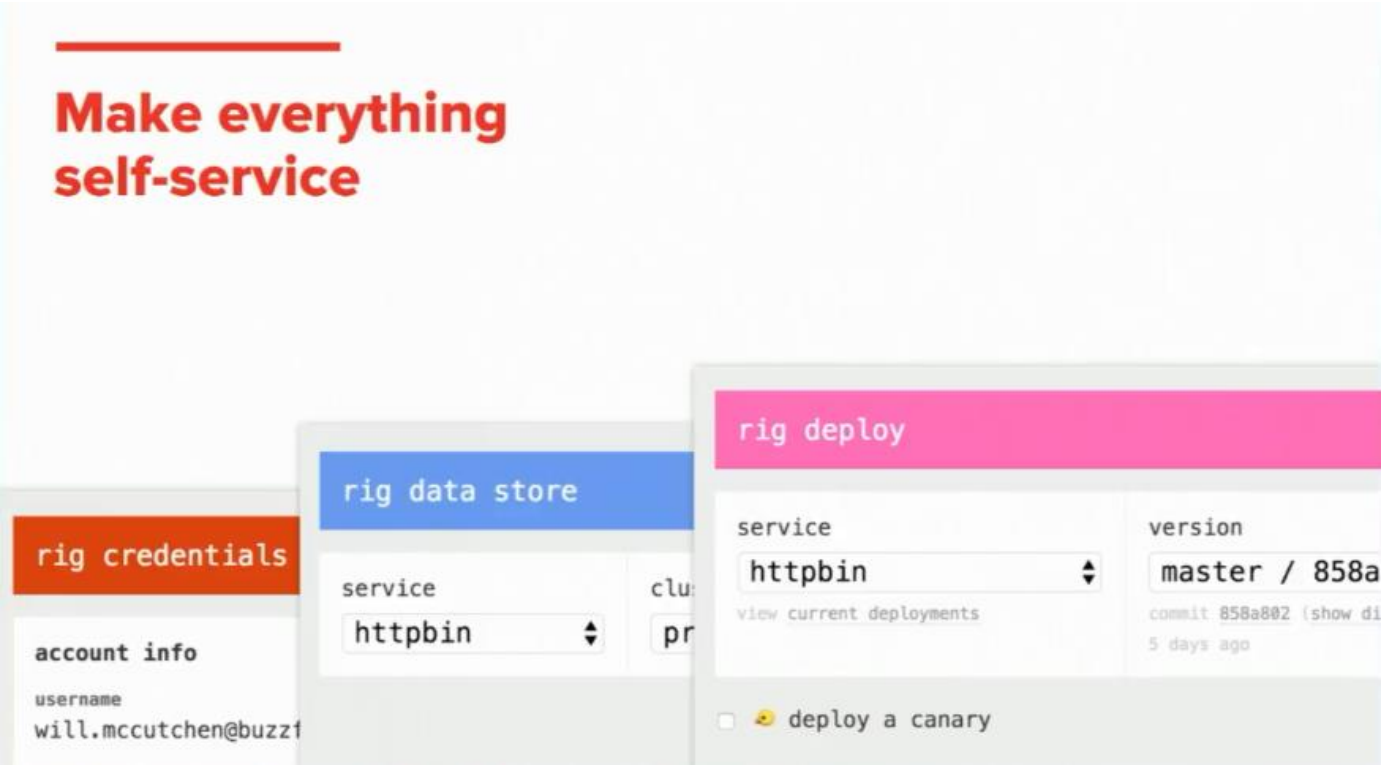
All rig services are in a single monolithic Git repo, builder gets the code and builds the Docker container and pushes it to ECR, then the user gets to provide 3 parameters, the service name, version, and the target cluster name, the system then goes ahead and deploys the code to ECS so that the user can see and use the app in the target cluster environment.



We have developed abstractions that are being enforced generally, like a rig service is a collection of the service definition that defines the structure of the service in a service.yml file, a configuration can also be created using the base config.yml file shown above



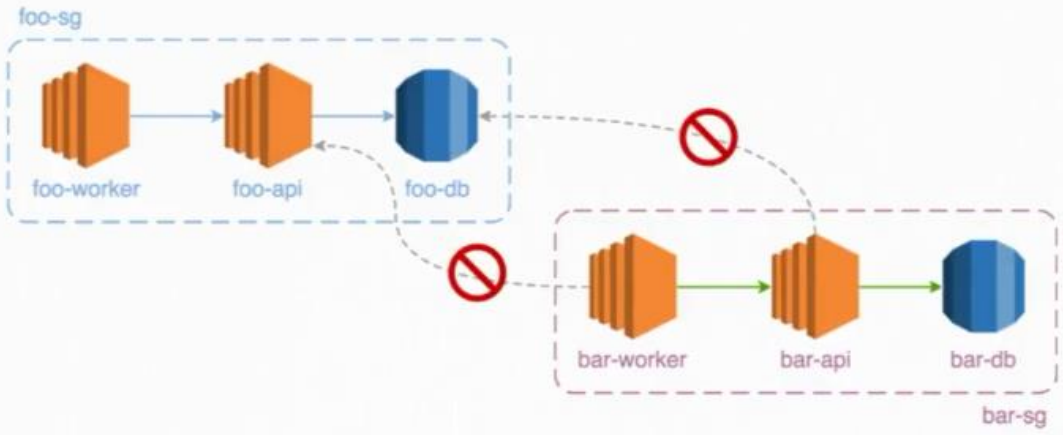
This is what a rig cluster looks like, we build immutable AMIs for all the ASGs that we create for the ECS services we run based on CloudWatch metrics. Every service exposes a network interface that uses a LB and a deterministic Route53 route



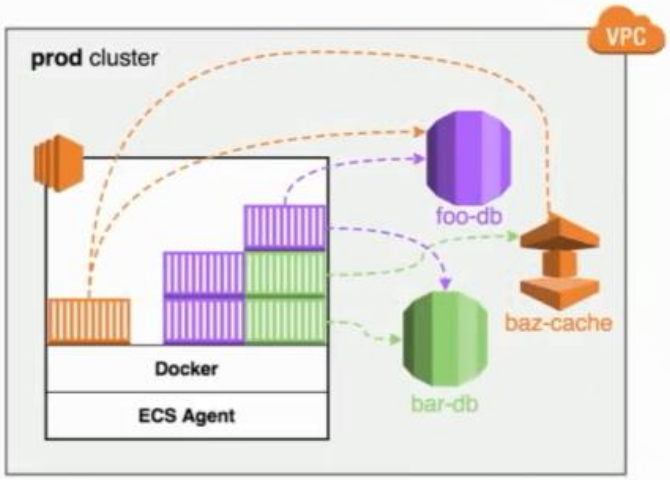
We also created self service UI apps that allows engineers to deploy their apps, configure datastores, credentials for user onboarding.

Some challenges

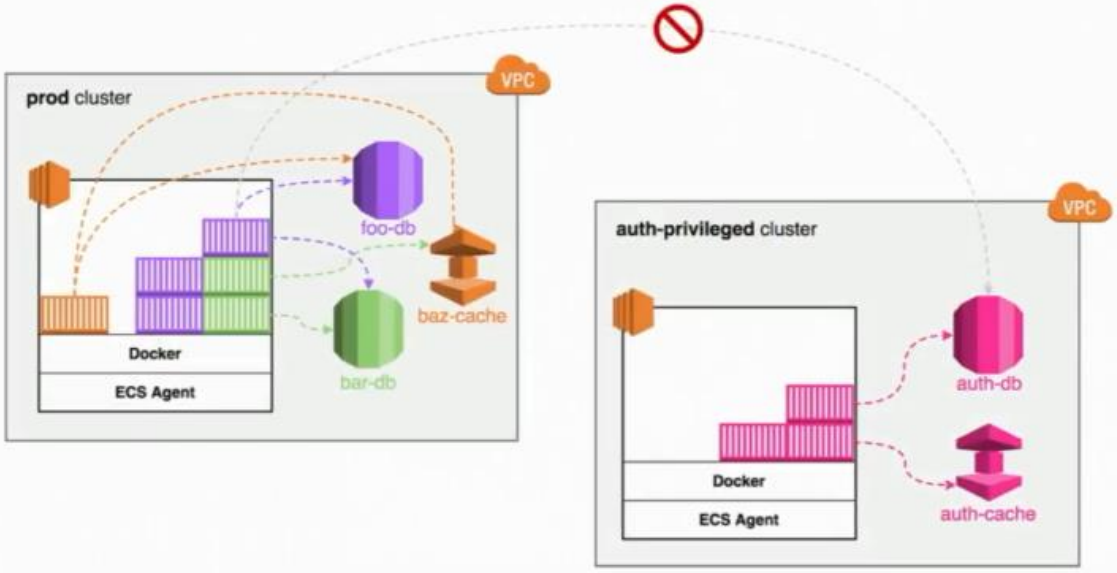
The good old days

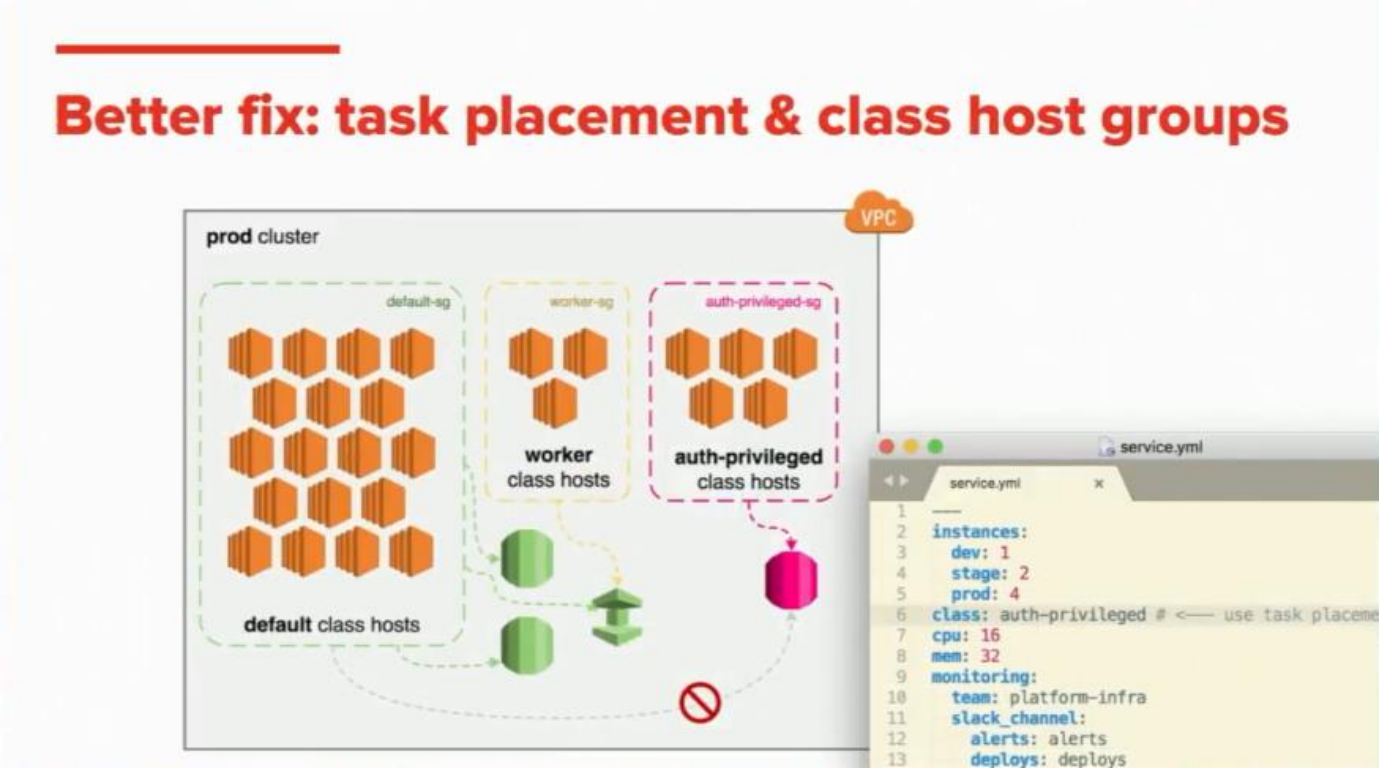


On ECS, everything runs everywhere



Quick fix: Isolated ECS clusters & VPCs





We have a lambda function that listens to the ECS cluster for things going on

Efficiency

What's next?