
AWS
re:Invent

GPS307

# Containers on AWS: Architecting software delivery platforms

Mikhail Shapirov

Sr. Partner Solutions Architect, Containers

AWS

AWS re:Invent

# Agenda

Enterprise setup – organization and environments

Day one – provisioning

Tenant onboarding

Workload onboarding

Operational strategies (accounts and regions)

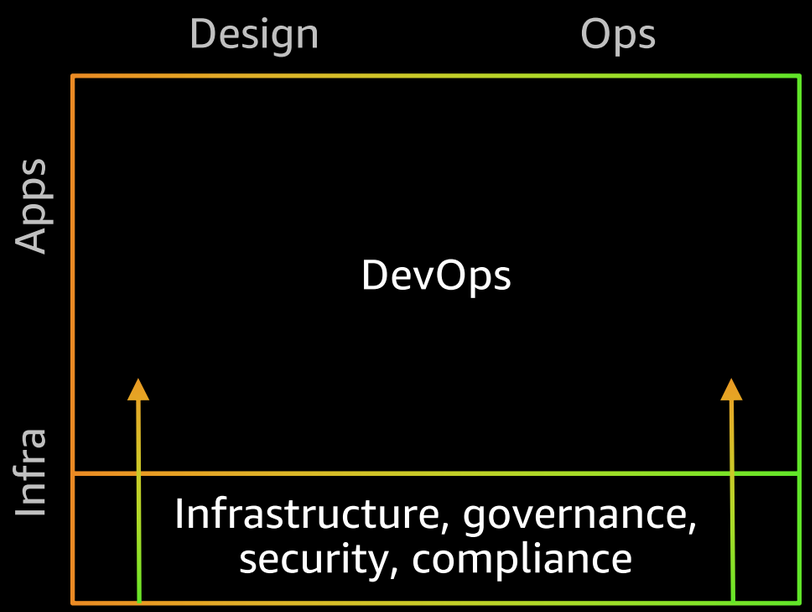
What is next?

# Enterprise DevOps

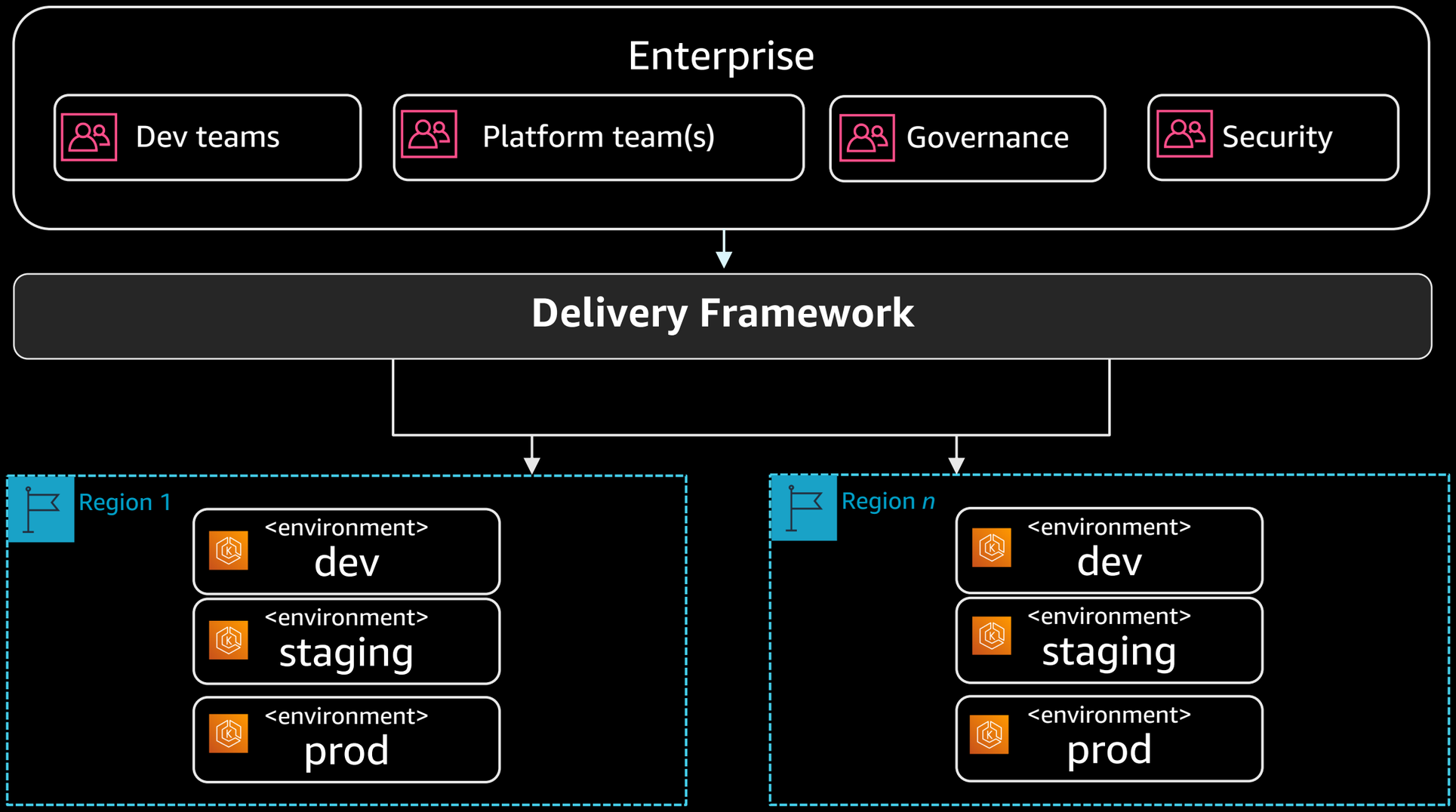
Traditional IT

Enterprise DevOps

DevOps (perception)



# Enterprise software delivery

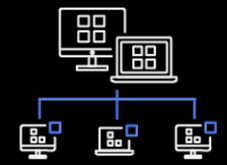


# Platform as a service (PaaS)

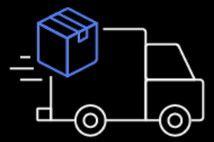
- Provider manages entire platform infrastructure
- You are abstracted from lower-level details of the environment

API wrapper

Workload manager



Customer endpoint



Deployment pipeline

Monitoring service

Metering service

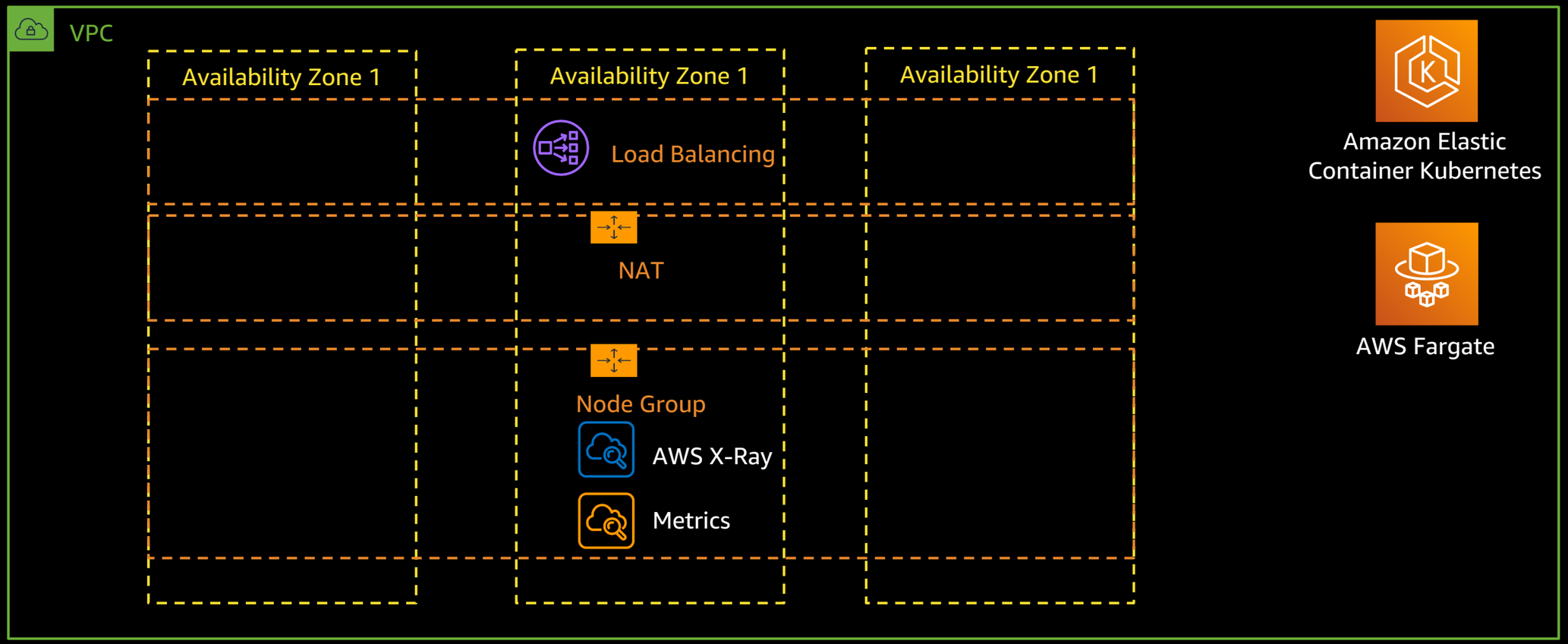
# Region scope

Amazon
API Gateway

Amazon Elastic
Container Registry

AWS App Mesh


Amazon Route 53






Amazon Elastic
Container Kubernetes

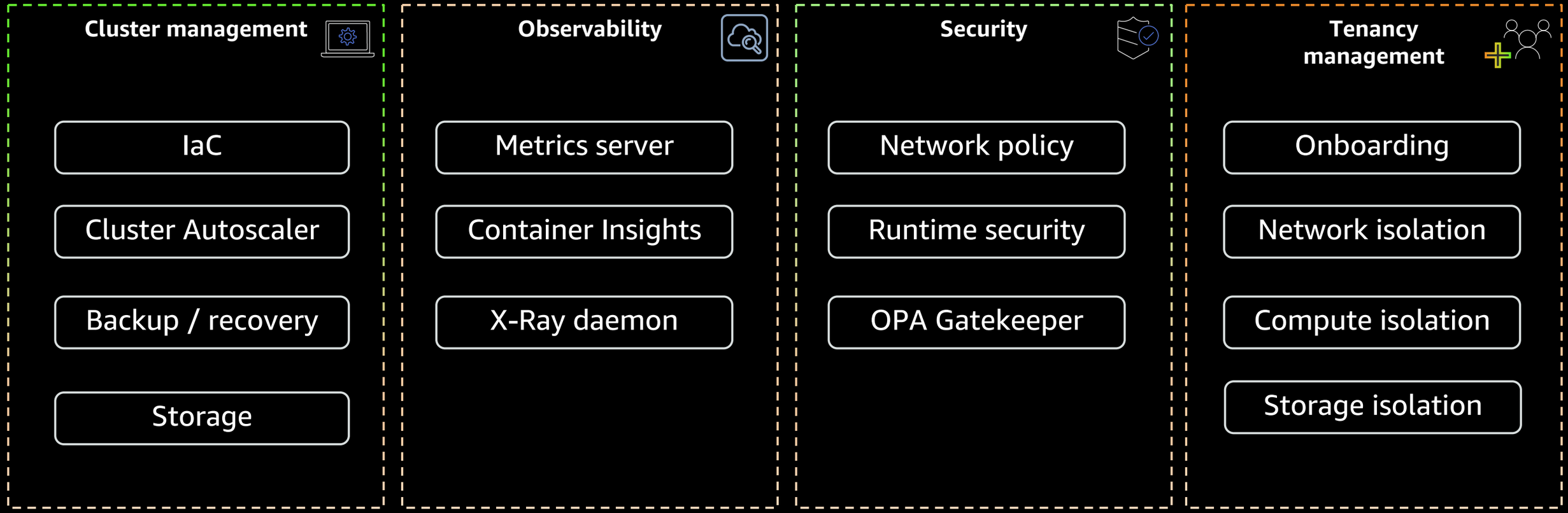


AWS Fargate

Amazon CloudWatch

AWS Certificate
Manager

# Cluster scope (EKS)



# Tenant (team) scope

Tenant-specific (Team = Namespace)

## Build

Source

- CodeCommit
- GitHub
- Bitbucket

CI

- AWS CodePipeline
- Tekton

## Deploy

- Resource control
- Ingress control
- Deployment control (blue / green, canary)

## Support

- CloudWatch
- Access to X-Ray
- Cluster Access (read-only)

# Day one: Provisioning

# Infrastructure as code (IaC)

- Actual code
- Define stack
- Create VPC
- Create cluster
- Create node group(s)

```
export class CdkEksBlueprintStack extends cdk.Stack {
…
  const vpc = new ec2.Vpc(this, clusterName + "-vpc");

  const cluster = new eks.Cluster(this, clusterName, {
    vpc: vpc,
    clusterName: clusterName,
    outputClusterName: true,
    defaultCapacity: 0,
    version: eks.KubernetesVersion.V1_18,
  });

  const ng = cluster.addNodegroupCapacity(ngName, {
    instanceType: new ec2.InstanceType(instanceType),
    minSize: 1,
    maxSize: 4
 });
```

# Cluster scope

- CNI and global network policy
- Metrics server (HPA)
- Cluster Autoscaler
- CloudWatch Container Insights
- NGINX Ingress Controller
- ArgoCD
- Many more

```
const addOns: Array<ClusterAddOn> = [
  new CalicoNetworkPolicyAddon,
  new MetricsServerAddon,
  new ClusterAutoScalerAddOn,
  new ContainerInsightsAddOn,
  new NginxAddon,
  new FluxCDAddon,
  new ArgoCDAddOn,
...
];

for(let addOn of addOns) { // strict order
  addOn.deploy(this);
}
```

# Tenant onboarding

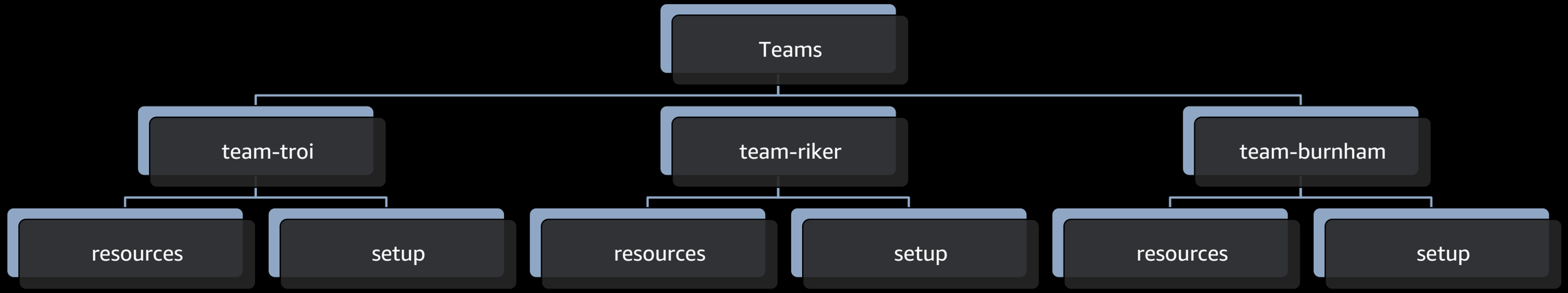
AWS re:Invent

# Tenant onboarding

- Create Namespace
- Set up network ingress / egress rules
- Set quotas
- Service accounts and IRSA
- Managed services setup
- Other setup

```
export class TeamTroiSetup implements TeamSetup {
  setup(cluster: eks.Cluster, stack: cdk.Stack) {
    const namespace = cluster.addManifest(this.teamName, {
      apiVersion: 'v1',
      kind: 'Namespace',
      metadata: { name: this.teamName }
    });

    this.setupNamespacePolicies(cluster);

    const sa = cluster.addServiceAccount('inf-backend',
       { name: 'inf-backend', namespace: this.teamName});
    sa.node.addDependency(namespace);
    const bucket = new s3.Bucket(stack, 'inf-inbox');
    bucket.grantReadWrite(sa);
…
  }
}
```

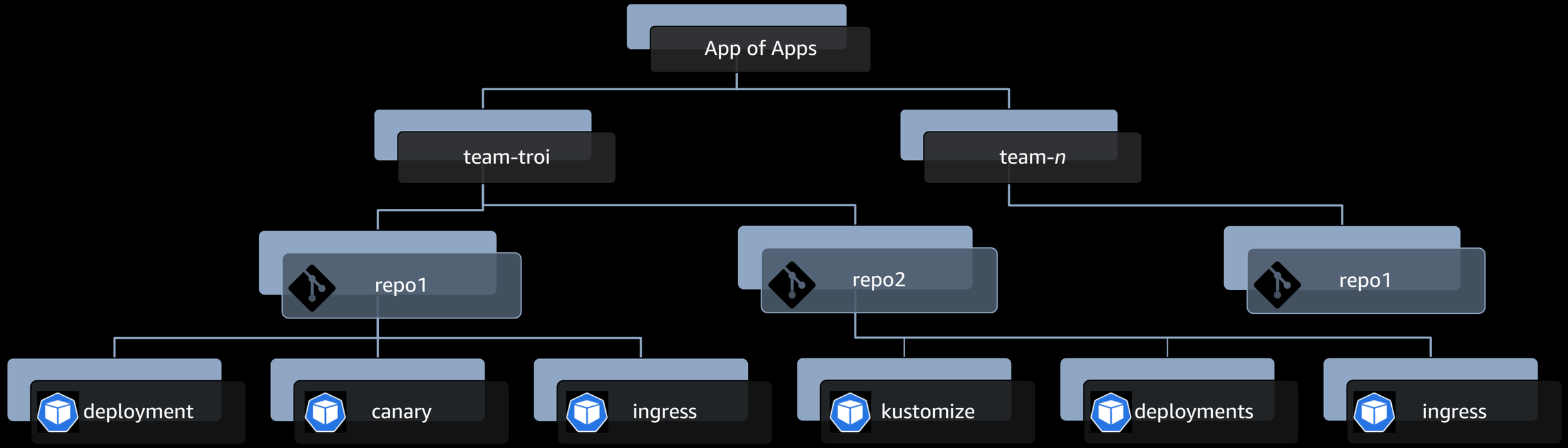
# Tenant onboarding




```
const teams: Array<TeamSetup> = [
  new TeamTroiSetup(),
  new TeamRikerSetup(),
  new TeamBurnhamSetup(),
...
];

teams.forEach(setup => setup.setup(cluster, this));
```

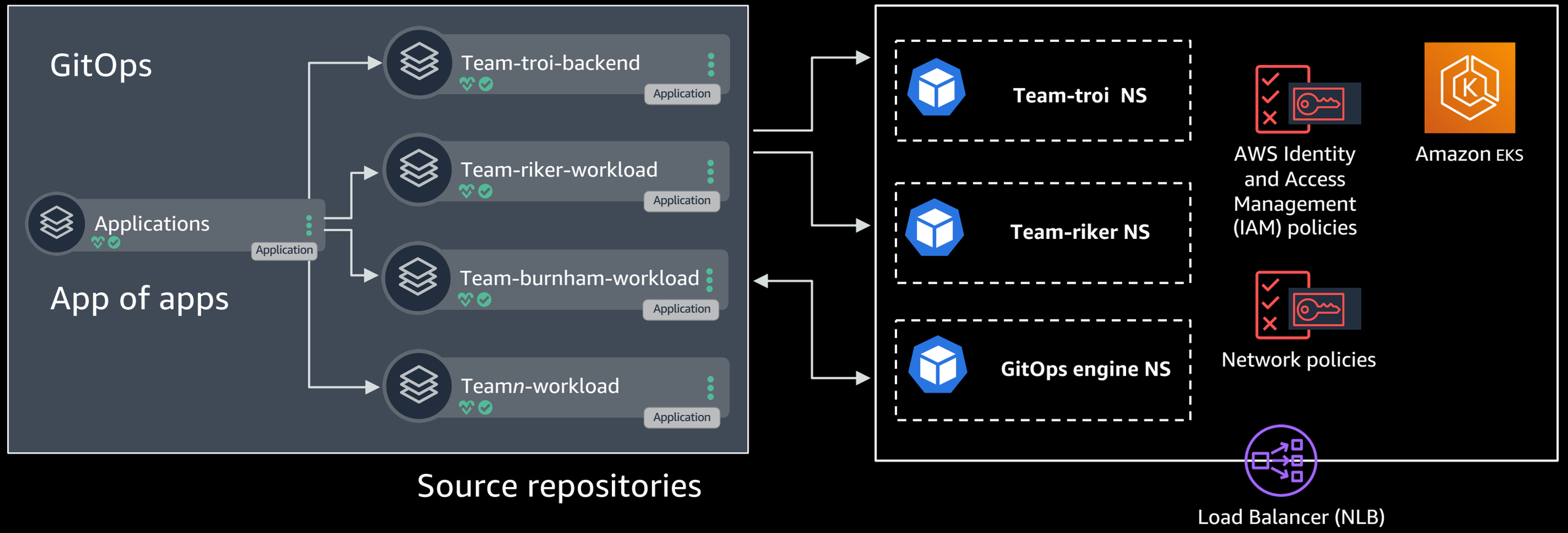
# Tenant workloads

Multi-tenant GitOps with ArgoCD



# Workload onboarding

## Cluster bootstrapping



# Bootstrapping: How?

1. Define apps using Kubernetes DSL (CRD) in single platform app repository

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: inf-backend-service
. . .
spec:
  destination:
    namespace: team-troi
  source:
    repoURL: git@{{troi-repo}}}
    path: team-troi
...
```

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: authorization-service
. . .
spec:
  destination:
    namespace: team-riker
  source:
    repoURL: git@{{riker-repo}}}
    path: team-riker
...
```

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: inventory-service
. . .
spec:
  destination:
    namespace: team-burnham
  source:
    repoURL: git@{{burnham-repo}}}
    path: team-burnham
...
```

2. Bootstrap

```
$ argocd app create apps --repo ${APP_REPO} …
$ argocd app sync apps
```

# Deployment strategies

| Capability | Argo Rollouts | Flagger | Service Mesh Direct | Kubernetes Direct |
|---|---|---|---|---|
| Blue / green | Yes | Yes | Yes | Yes |
| Canary | Yes | Yes | Yes | No |
| Metrics-driven decisions | Automatic | Automatic | Manual | Manual |
| Rollback | Automatic | Automatic | Manual | Manual |
| Analysis | Built-in | Built-in | Manual | Manual |

# Progressive delivery with Flagger

```yaml
apiVersion: flagger.app/v1beta1
kind: Canary
metadata:
  name: inf-backend
spec:
  # service mesh provider can be: kubernetes, istio, appmesh, nginx, gloo
  provider: nginx
  # deployment reference
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: inf-backend
  # the maximum time in seconds before rollback
  progressDeadlineSeconds: 60
  # HPA reference (optional)
  autoscalerRef:
    apiVersion: autoscaling/v2beta1
    kind: HorizontalPodAutoscaler
    name: inf-backend
```

# Multi-cluster, multi-region, and multi-account

# Multi-cluster, multi-region

Deploy a single dev cluster

```
new CdkEksBlueprintStack(app, 'dev-stack’,
    "dev-cluster");

$ cdk deploy dev-stack
```

Deploy a few staging clusters to multiple regions

```
new CdkEksBlueprintStack(app, 'east-staging-stack’,
“staging-cluster", { env: {region: 'us-east-2’}});


new CdkEksBlueprintStack(app, ‘west-staging-stack’,
“staging-cluster", { env: {region: ‘us-west-2’}});
```

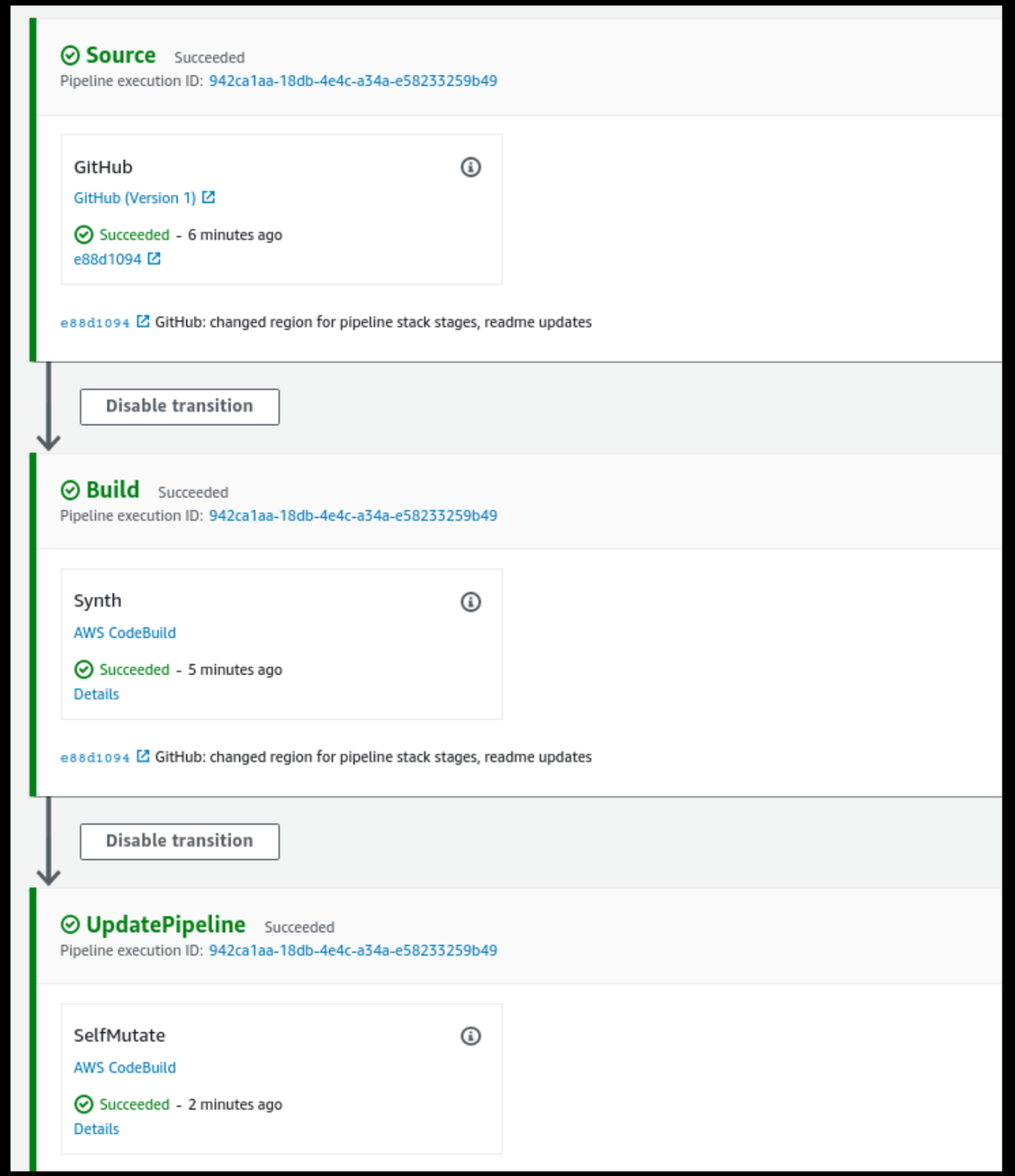
# Multi-account

Split dev and prod accounts

```
new CdkEksBlueprintStack(app, 'east-dev-stack,
{
  env: {
    region: 'us-east-2’,
    account: ‘DEV_ACCOUNT’
  }
});


new CdkEksBlueprintStack(app, 'east-prod-stack,
{
  env: {
    region: 'us-east-2’,
    account: ‘PROD_ACCOUNT’
  }
});
```

# IaC pipeline

```
// Do this as many times as necessary
// with any account and region
pipeline.addApplicationStage(
    new FactoryApplication(this, 'dev', {
      env: {
         account: '${DEV_ACCOUNT}',
         region: 'us-east-2',

      }
}));

pipeline.addApplicationStage(
    new FactoryApplication(this, 'staging', {
      env: {
        account: '${ST_ACCOUNT}',
        region: 'us-east-2'
      }
}));
```



# References


- https://github.com/shapirov103/cdk-eks-blueprint
- https://github.com/shapirov103/argo-apps
- https://argoproj.github.io/argo-cd/
- https://github.com/weaveworks/flagger
- https://argoproj.github.io/argo-rollouts
- https://d1.awsstatic.com/events/reinvent/2019/Architecting_multi-tenant_PaaS_offerings_with_Amazon_EKS_GPSTEC337.pdf
- https://www.slideshare.net/AmazonWebServices/enterprise-devops-patterns-of-efficiency-ent311r1-aws-reinvent-2018
- https://www.gartner.com/en/documents/3989064/hype-cycle-for-platform-as-a-service-2020
- https://searchcloudcomputing.techtarget.com/opinion/Why-the-PaaS-market-failed-to-live-up-to-the-hype

# Thank you!

AWS re:Invent

# Please complete the session survey

AWS re:Invent