AWS

SUMMIT

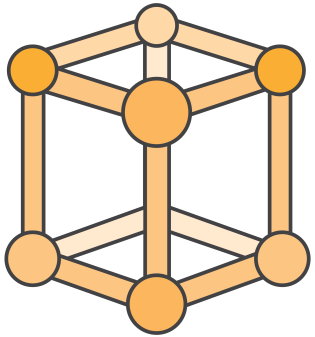
Advanced Task Scheduling with Amazon ECS and Blox

Sascha Möllering, Solutions Architect, @sascha242

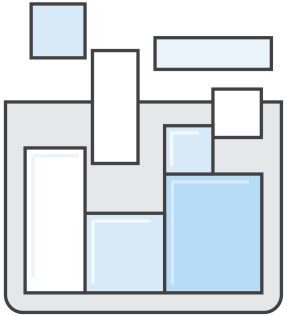
May 18th, 2017

amazon web services

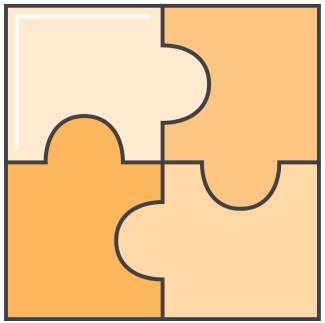
What is Amazon ECS?



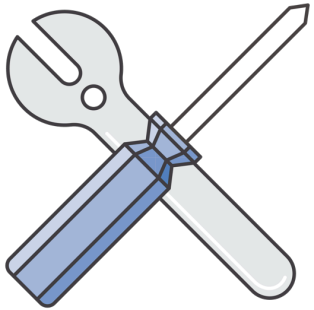
Scalable Container Management



Flexible Container Placement

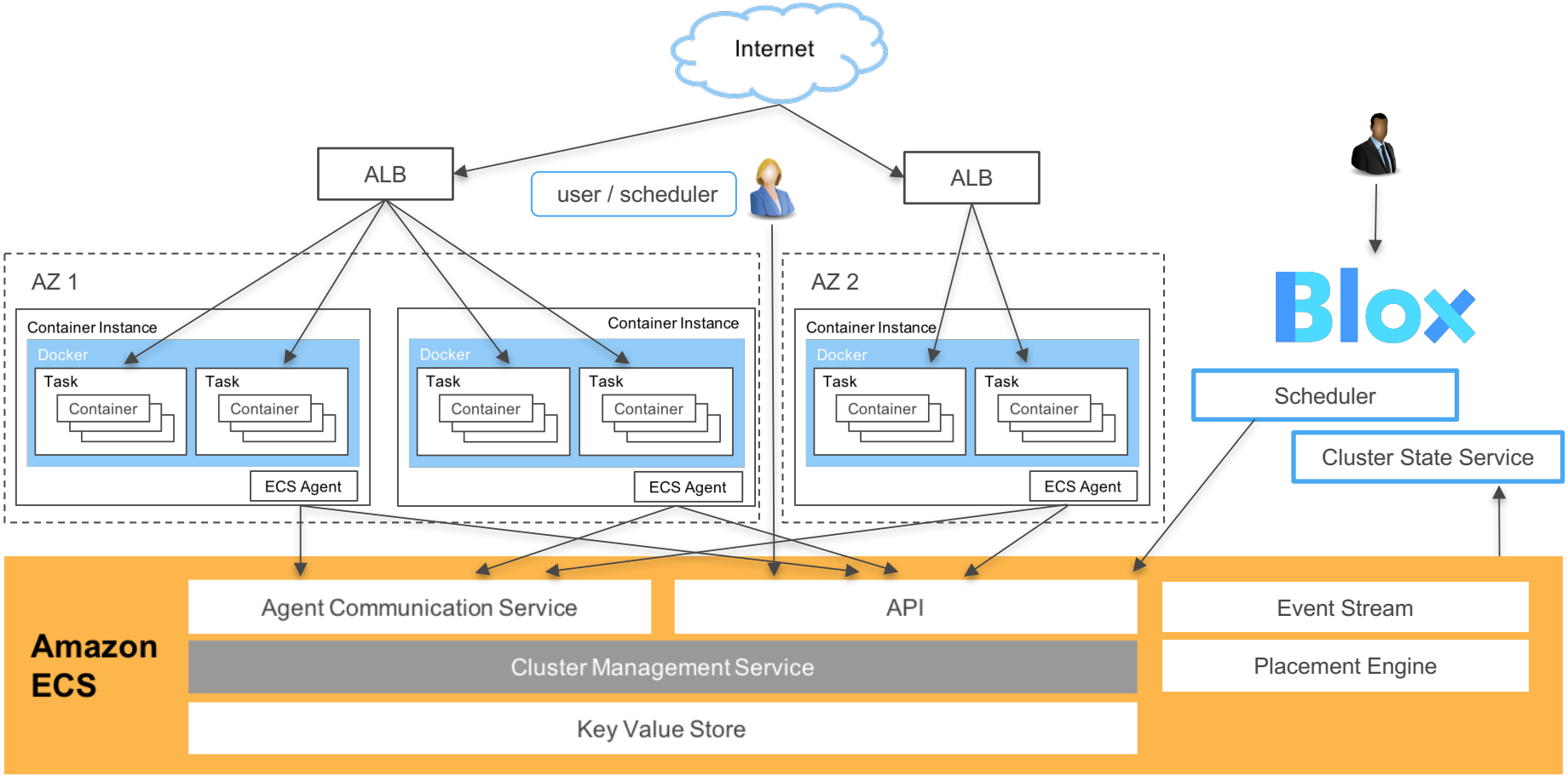


AWS Platform Integration



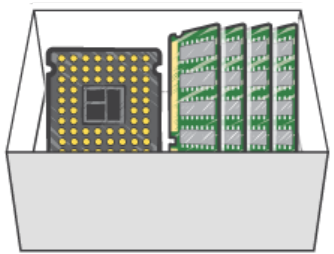
Extensible by Design

Amazon ECS: Under the Hood



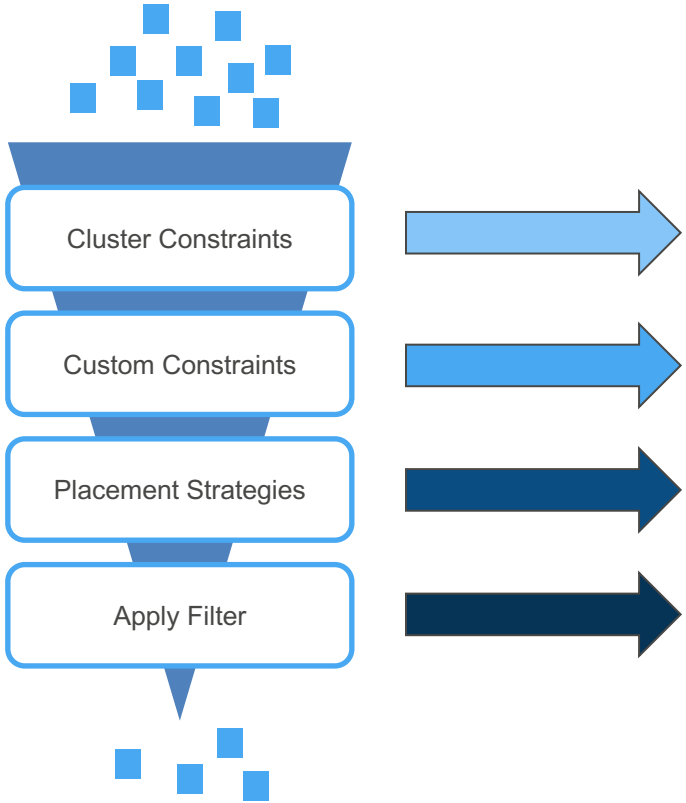
Task Placement Engine

New Placement Constraints & Attributes



	Name	Example
✓	AMI ID	attribute:ecs.ami-id == ami-eca289fb
✓	Availability Zone	attribute:ecs.availability-zone == us-east-1a
✓	Instance Type	attribute:ecs.instance-type == t2.small
✓	Distinct Instances	type="distinctInstance"
✓	Custom	attribute:stack == prod

Anatomy of Task Placement



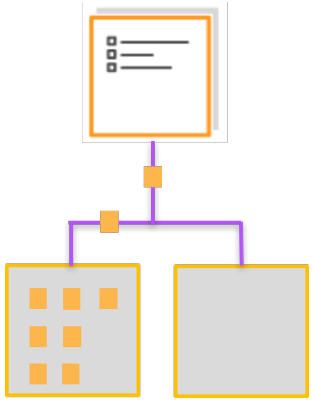
Satisfy CPU, memory, and port requirements

Filter for location, instance-type, AMI, or custom attribute constraints

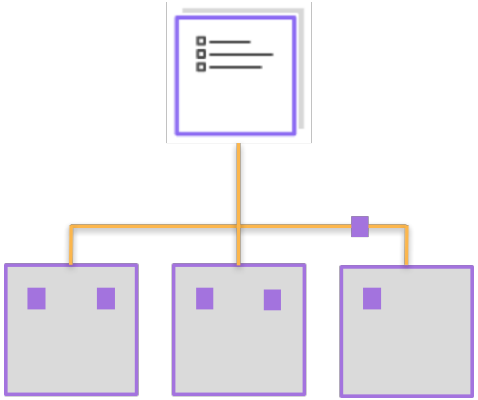
Identify instances that meet spread or binpack placement strategy

Select final container instances for placement

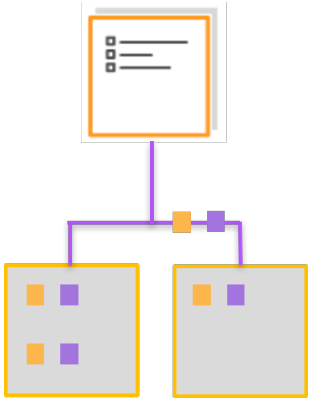
Supported Placement Strategies



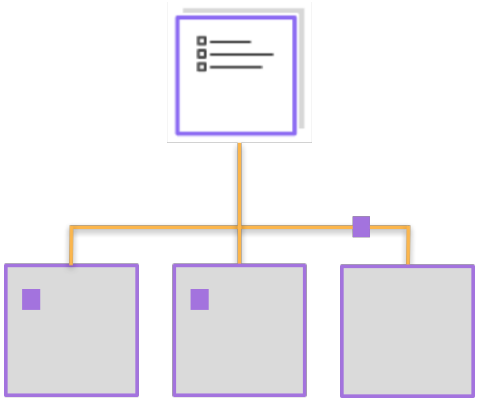
Binpacking



Spread

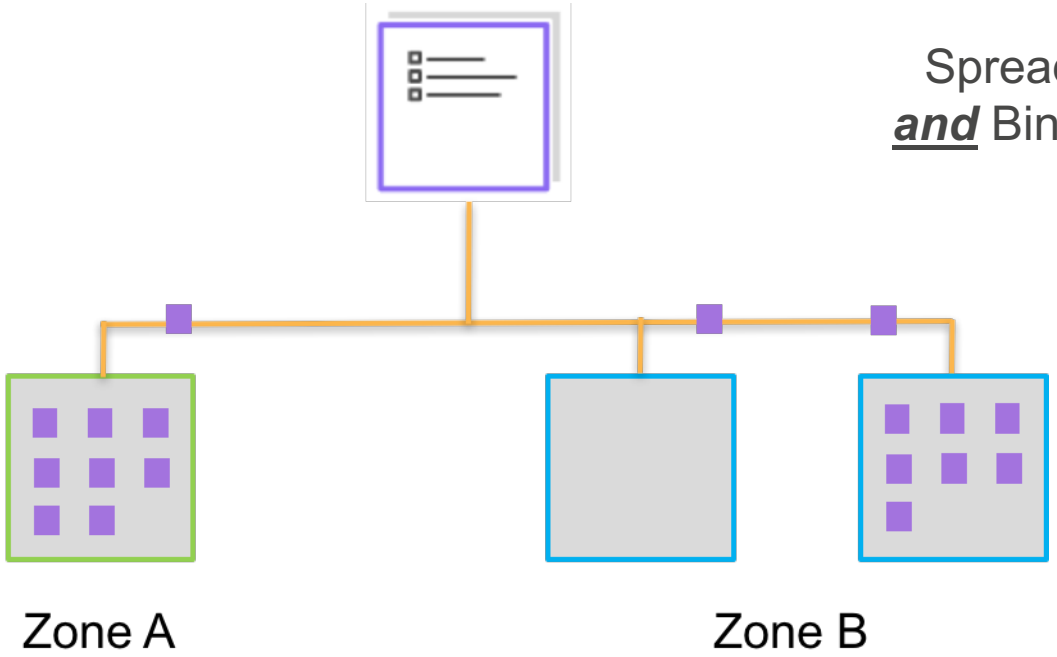


Affinity



Distinct Instance

Placement Strategy Chaining



Spread tasks across Zones
and Binpack within each Zone

New Cluster Query Language

Filtering: Match on Instance Family or Type

```
aws ecs list-container-instances --cluster ecs-demo --filter "attribute:ecs.instance-type matches t2.*"
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:123456789000:container-instance/3ced5d42-537c-40b4-9551-b9022cc13b78",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/442d988b-4b00-40bf-85ae-34e0819454f2",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/5de11ede-6c22-411e-a469-8301eeebae0f",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/6bfb12c8-1c3c-4d4a-976c-ce3c2c79b031",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/7eb87781-abab-4a6a-9a0d-602a4da59549",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/af8d48ba-73c4-409a-b40f-66596aa86c5d",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/b5c08e3e-bd25-4ec9-9a88-ce1d53640542",
    ]
}
```

```
aws ecs list-container-instances --cluster ecs-demo --filter "attribute:ecs.instance-type matches t2.small"
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:123456789000:container-instance/af8d48ba-73c4-409a-b40f-66596aa86c5d",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/b5c08e3e-bd25-4ec9-9a88-ce1d53640542",
    ]
}
```

Filtering: Match on Availability Zone

```
aws ecs list-container-instances --cluster ecs-demo --filter "attribute:ecs.availability-zone matches us-east-1.*"
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:123456789000:container-instance/3ced5d42-537c-40b4-9551-b9022cc13b78",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/442d988b-4b00-40bf-85ae-34e0819454f2",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/5de11ede-6c22-411e-a469-8301eeebae0f",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/6bfb12c8-1c3c-4d4a-976c-ce3c2c79b031",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/7eb87781-abab-4a6a-9a0d-602a4da59549",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/af8d48ba-73c4-409a-b40f-66596aa86c5d",
    ]
}
```

```
aws ecs list-container-instances --cluster ecs-demo --filter "attribute:ecs.availability-zone == us-east-1a"
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:123456789000:container-instance/3ced5d42-537c-40b4-9551-b9022cc13b78",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/442d988b-4b00-40bf-85ae-34e0819454f2",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/5de11ede-6c22-411e-a469-8301eeebae0f",
    ]
}
```

Filtering: Match on Multiple Expressions

```
aws ecs list-container-instances --cluster ecs-demo --filter "attributes:ecs.instance-type matches t2.* and
attribute:ecs.availability-zone == us-east-1a"
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:123456789000:container-instance/3ced5d42-537c-40b4-9551-b9022cc13b78",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/442d988b-4b00-40bf-85ae-34e0819454f2",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/5de11ede-6c22-411e-a469-8301eeebae0f",
    ]
}
```

```
aws ecs list-container-instances --cluster ecs-demo --filter "(attribute:ecs.instance-type in [t2.small, t2.medium]
or attribute:ecs.instance-type matches g2.*) and attribute:ecs.availability-zone != us-east-1d"
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:123456789000:container-instance/3ced5d42-537c-40b4-9551-b9022cc13b78",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/442d988b-4b00-40bf-85ae-34e0819454f2",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/5de11ede-6c22-411e-a469-8301eeebae0f",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/6bfb12c8-1c3c-4d4a-976c-ce3c2c79b031",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/7eb87781-abab-4a6a-9a0d-602a4da59549",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/d45f5b92-4faa-44a9-a9a7-2d744566e510",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/f3d92b17-7d95-4cff-b623-390e871c6b60",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/f7858158-5806-4d8d-82ea-f0eb4680e6cf",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/fc751042-590a-440d-90a7-e8ebce02d234"
    ]
}
```

Filtering: Match on Custom Attributes

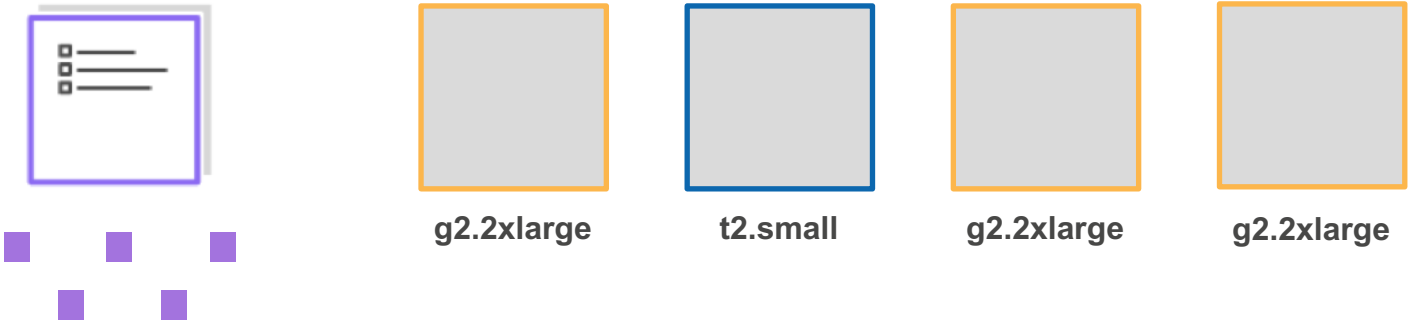
```
aws ecs associate-attributes --cluster ecs-demo --target-id 3ced5d42-537c-40b4-9551-b9022cc13b78
--target-type container-instance --attribute name=stack, value=prod
```

```
aws ecs list-container-instances --cluster ecs-demo --filter "attribute:stack != prod"
{
    "containerInstanceArns": [
        "arn:aws:ecs:us-east-1:123456789000:container-instance/442d988b-4b00-40bf-85ae-34e0819454f2",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/5de11ede-6c22-411e-a469-8301eeebae0f",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/6bfb12c8-1c3c-4d4a-976c-ce3c2c79b031",
        "arn:aws:ecs:us-east-1:123456789000:container-instance/af8d48ba-73c4-409a-b40f-66596aa86c5d",
    ]
}
```

Task Placement Examples

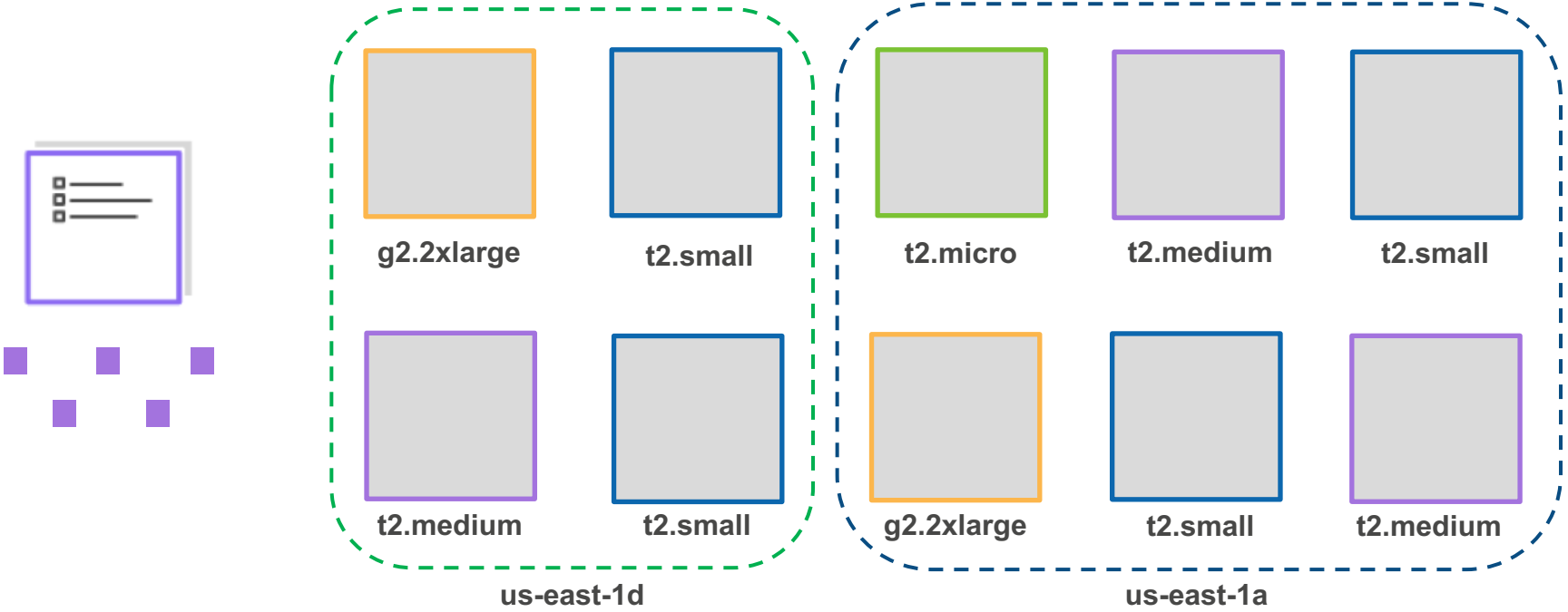
Placement: Targeting Instance Type

```
aws ecs run-task --cluster ecs-demo --task-definition myapp --count 5 --placement-constraints
type="memberOf",expression="attribute:ecs.instance-type == g2.2xlarge"
```



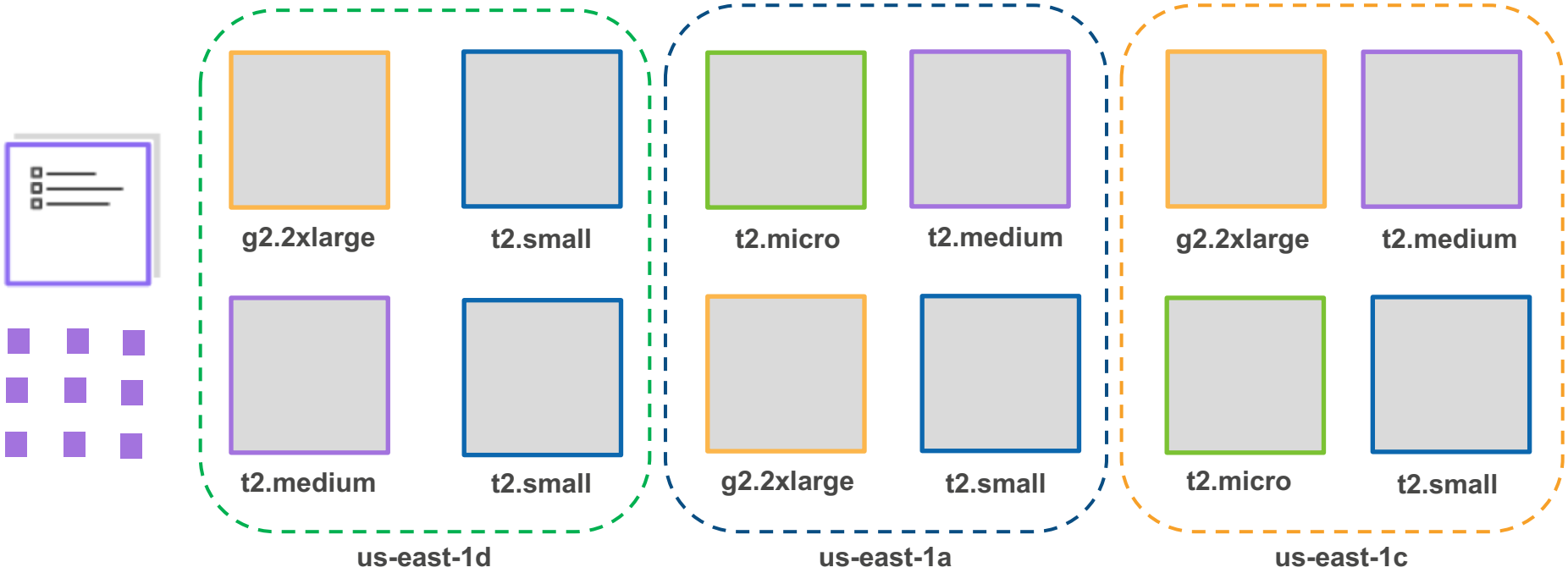
Placement: Targeting Instance Type & Zone

```
aws ecs run-task --cluster ecs-demo --task-definition myapp --count 5 --placement-constraints
type="memberOf",expression="(attribute:ecs.instance-type == t2.small or
attribute:ecs.instance-type == t2.medium) and attribute:ecs.availability-zone != us-east-1d"
```



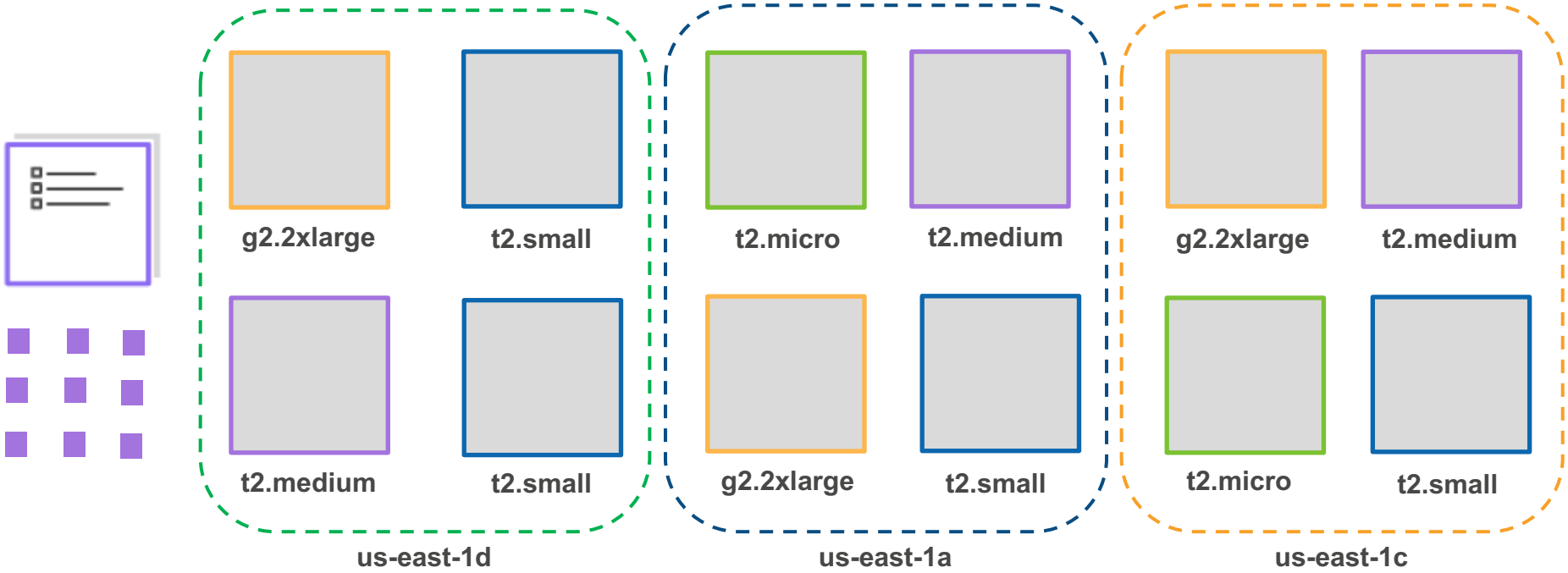
Placement: Availability Zone Spread

```
aws ecs run-task --cluster ecs-demo --task-definition myapp --count 9 --placement-strategy
type="spread",field="attribute:ecs.availability-zone"
```



Placement: Spread across Zone and Binpack

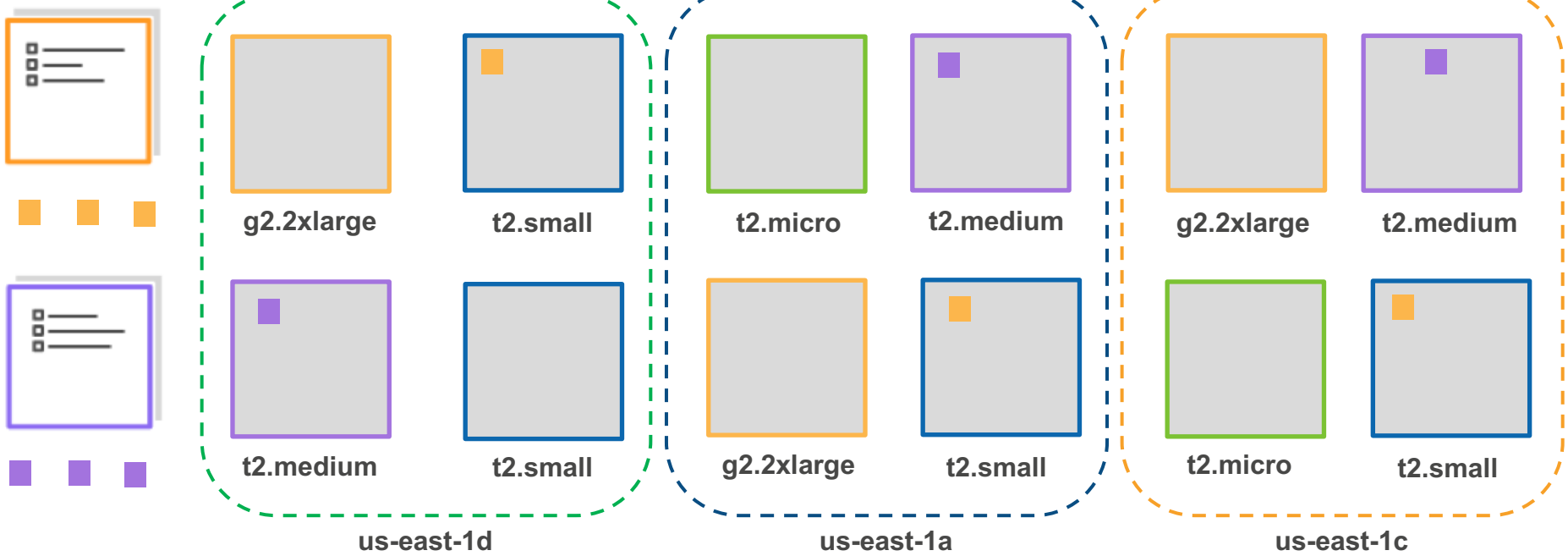
```
aws ecs run-task --cluster ecs-demo --task-definition myapp --count 9 --placement-strategy
type="spread",field="attribute:ecs.availability-zone" type="binpack",field="memory"
```



Placement: Affinity and Anti-Affinity

```
aws ecs run-task --count 3 --cluster ecs-demo --task-definition myapp --group webserver --placement-constraints
type=memberOf,expression="task:group == webserver"

aws ecs run-task --count 3 --cluster ecs-demo --task-definition mydb --group dbserver --placement-constraints
type=memberOf,expression="not(task:group == webserver)"
```



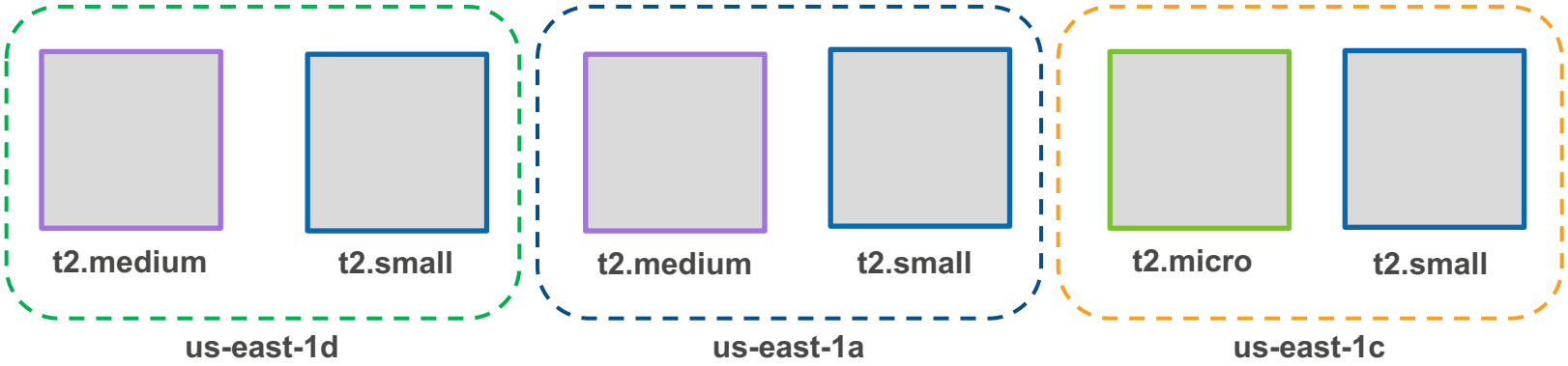
Running a Service

```
{
    "cluster": "ecs-demo",
    "serviceName": "my-service",
    "taskDefinition": "my-app",
    "desiredCount": 10,
    "placementConstraints": [
        {
            "type": "memberOf",
            "expression": "attribute:ecs.instance-type matches t2.*"
        }
    ],
    "placementStrategy": [
        {
            "type": "spread",
            "field": "attribute:ecs.availability-zone"
        },
        {
            "type": "binpack",
            "field": "MEMORY"
        }
    ]
}
```

Placement: Multiple Services on a Cluster

```
aws ecs create-service --service-name srvc-binpk --cluster ecs-demo --task-definition myapp-binpk
--desired-count 5 --placement-strategy type="binpack",field="memory"
```

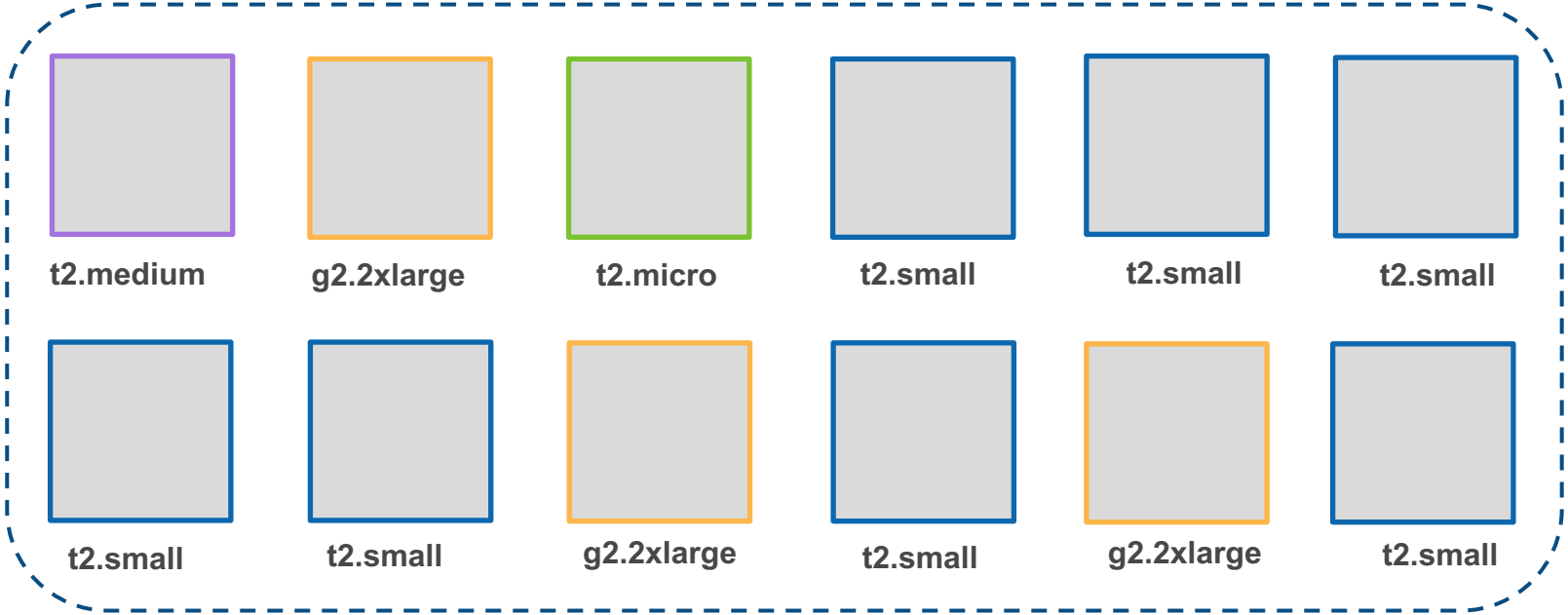
```
aws ecs create-service --service-name srvc-spread --cluster ecs-demo --task-definition myapp-spread
--desired-count 6 --placement-strategy type="spread",field="attribute:ecs.availability-zone"
```



Placement: Services – Distinct Instances

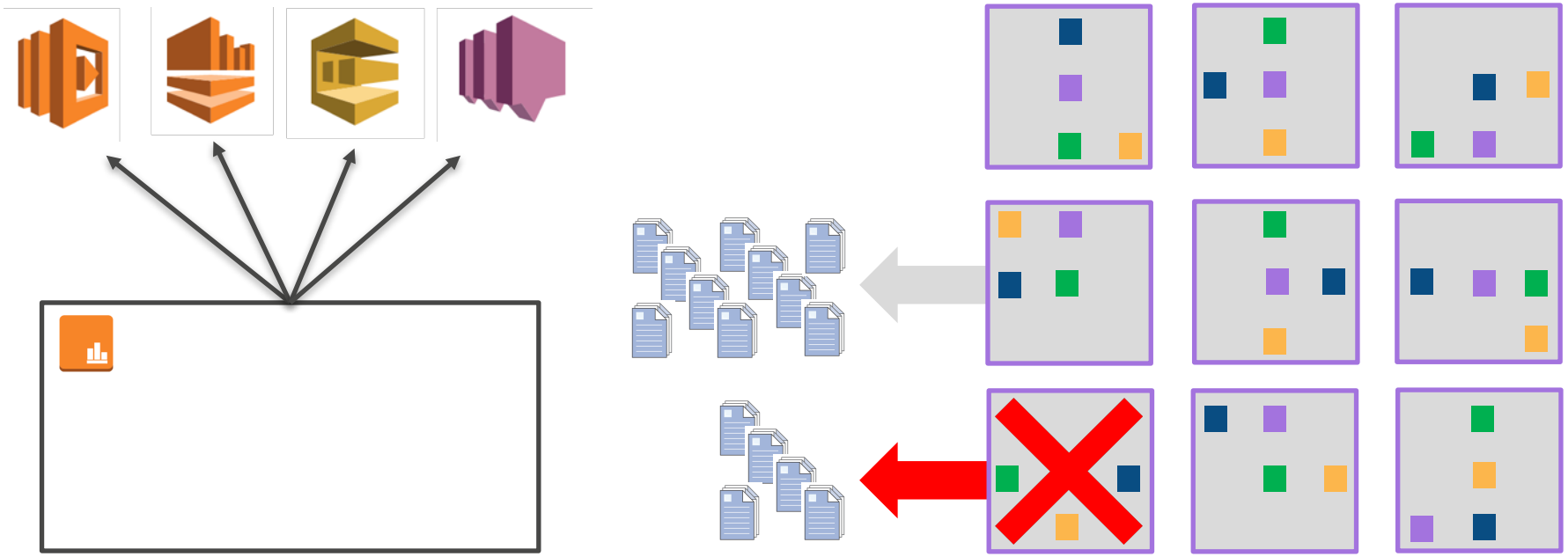
```
aws ecs create-service --service-name myapp-gpu --cluster ecs-demo --task-definition myapp-gpu --desired-count 3 --placement-constraints type="memberOf",expression="attribute:ecs.instance-type =~ g2.*" type="distinctInstance"
```

```
aws ecs create-service --service-name myapp --cluster ecs-demo --task-definition myapp --desired-count 9 --placement-constraints type="memberOf",expression="(attribute:ecs.instance-type == t2.small or attribute:ecs.instance-type == t2.medium)" type="distinctInstance"
```



Event Stream

Consuming Real-time Events



Three Steps to Getting Started with Events

Step 1: Create CWE Rule

```
> cat ecs-cw-rule.json
{
  "source": [
      "aws.ecs"
  ],
  "detail-type": [
      "ECS Task State Change",
      "ECS Container Instance State Change"
  ]
}
> aws events put-rule --name EventStreamRule --event-pattern
file:///path/ecs-cw-rule.json
```

Step 2: Create SNS Topic

```
> aws sns create-topic --name EventStreamTopic
```

Step 3: Put Events to SNS

```
> aws events put-targets --rule EventStreamRule --targets "Id
=EventStreamTarget,Arn=<topic_arn>"
```

Real-time Events with AWS Lambda

```
function postToSNS(event, context) {
     data = event["detail"]
     if (data["lastStatus"] == "STOPPED") {
          var sns_client = new AWS.SNS();
          sns_client.publish({
               Message: 'ECS task notification:\n\n' + JSON.stringify(data, null, 2),
TopicArn: sns_topic,
          }, function(err, data) {
               if (err) {
                    console.log(err.stack);
                    return;
               }
               context.done(null, 'Function Finished!');
          });
     } else {
          context.done(null, 'Function Finished!');
}}
```

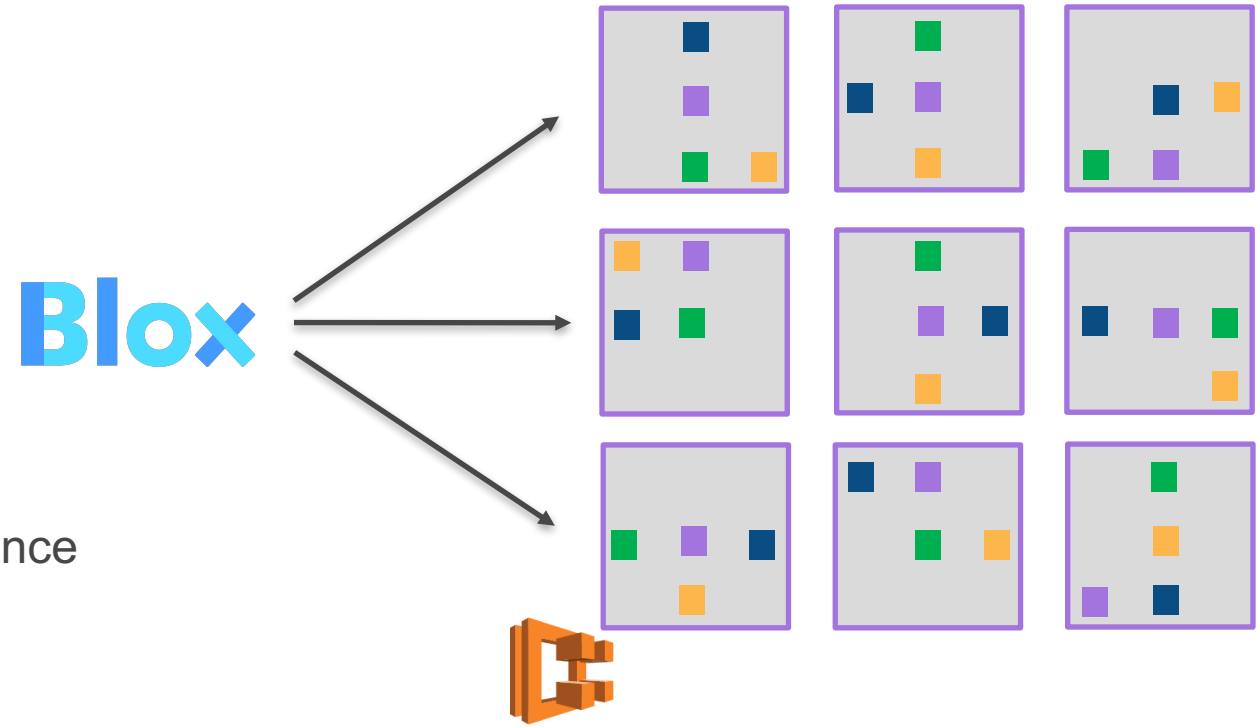
Introducing Blox

What is Blox?

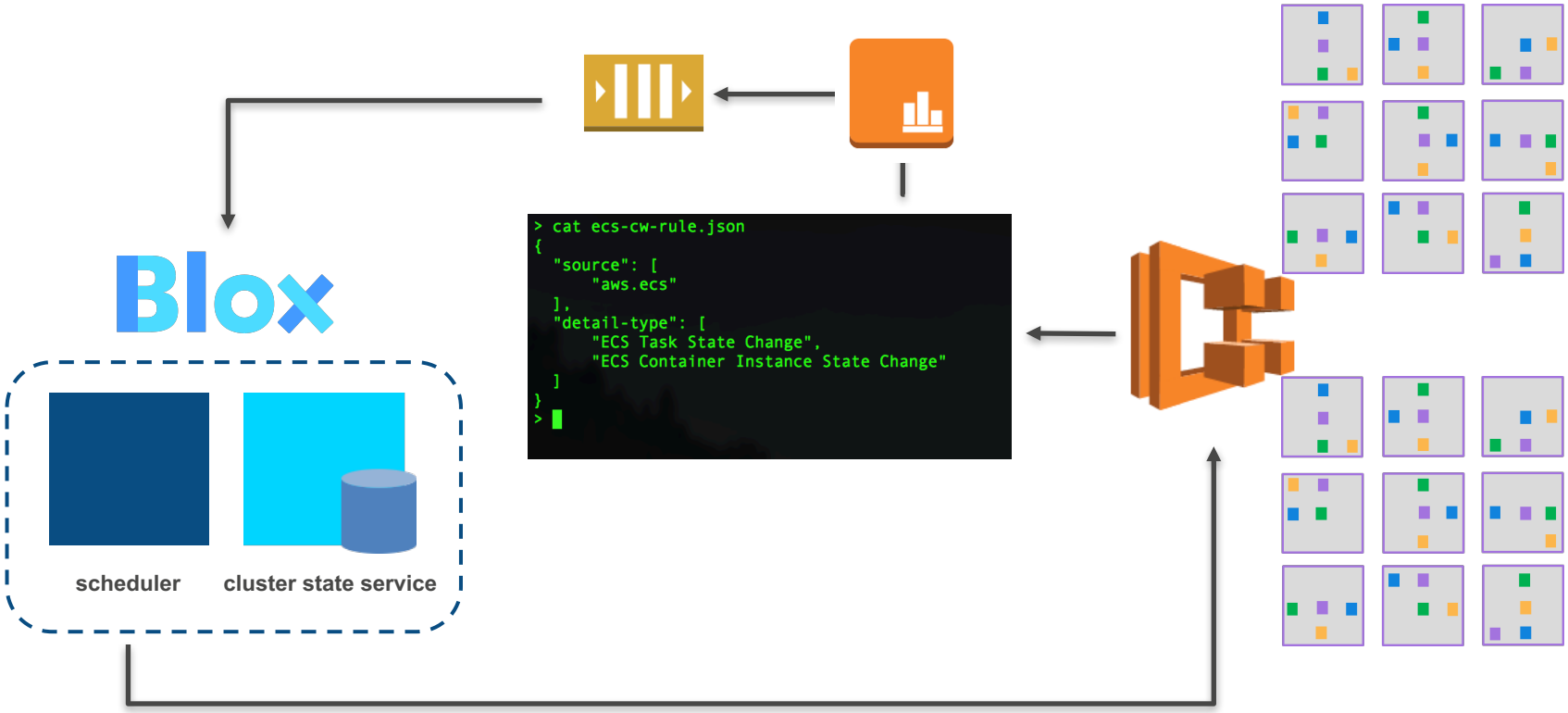
Choice

Control

Developer Experience



Building with Blox



Set Up Blox Locally

Step 1: git clone https://github.com/blox/blox.git

Step 2: deploy CloudFormation template to configure ECS event stream and SQS queue

```
> aws cloudformation create-stack --stack-name blox-local --template-body
file:///path/blox-template.json
```

Step 3: docker-compose up -d

```
> docker-compose ps
      Name                     Command               State                Ports
-------------------------------------------------------------------------------------------
conf_css_1          /cluster-state-service --b ...   Up      0.0.0.0:3000->3000/tcp, 80/tcp
conf_etcd_1         /usr/local/bin/etcd --data ...   Up      0.0.0.0:2379->2379/tcp, 2380/tcp
conf_scheduler_1    /daemon-scheduler --bind 0 ...   Up      0.0.0.0:2000->2000/tcp
>
```

Step 4: start using Blox locally

Swagger Spec: Cluster State Service

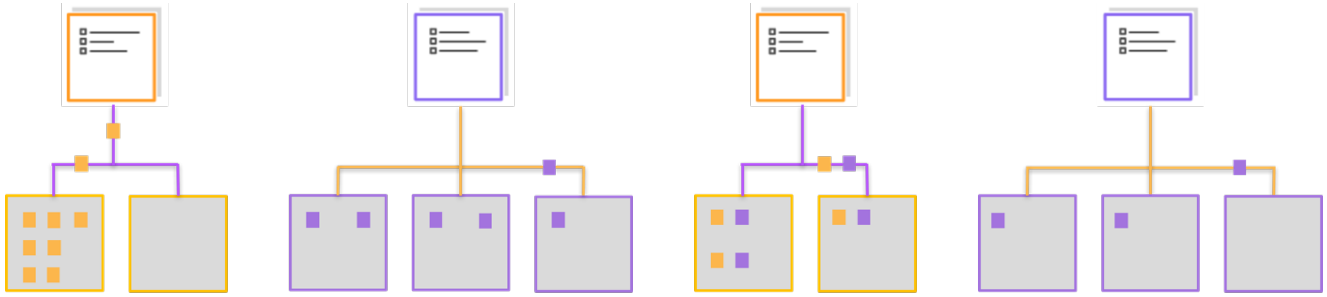
GET	/instances/{cluster}/{arn}
GET	/instances
GET	/stream/instances
GET	/tasks/{cluster}/{arn}
GET	/tasks
GET	/stream/tasks

Swagger Spec: Daemon Scheduler

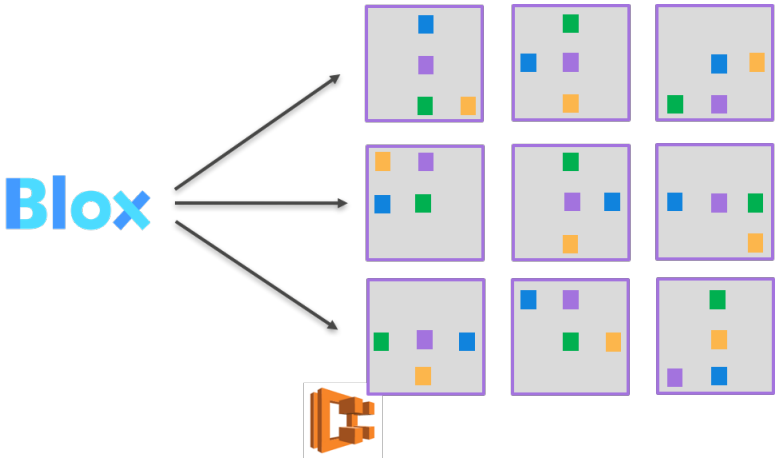
GET	/ping
POST	/environments
GET	/environments
GET	/environments/{name}
DELETE	/environments/{name}
POST	/environments/{name}/deployments
GET	/environments/{name}/deployments
GET	/environments/{name}/deployments/{id}

Summary

Summary



Binpacking Spread Affinity Distinct Instance



AWS

SUMMIT

Thank you!

amazon web services