BUILDING HYBRID MICROSERVICES ON EKS

Leon Jalfon

DevOps & Cloud Architect

SELA GROUP | MORE. By Sela Group

aws | FLOOR28 TEL-AVIV

About Sela Group & More

- SELA Group is a global company with over two decades of track record in development, training, and consulting.

- 10 Years in the cloud business – from reselling, onboarding, developing, training, optimizing, and mostly delivering complete cloud solutions

- MORE by Sela Group was built with one vision in mind: Delivering an holistic Multi Cloud service, from development to production.

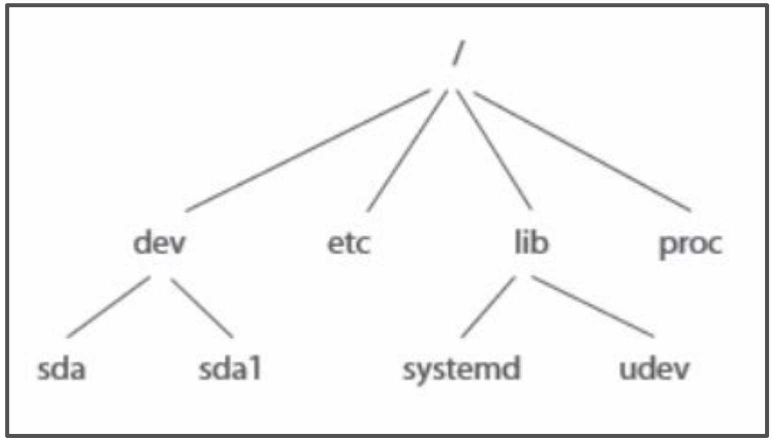
Agenda

- Introduction
- Case of Study Overview
- Implementation and Demos
- Summary
- Q&A

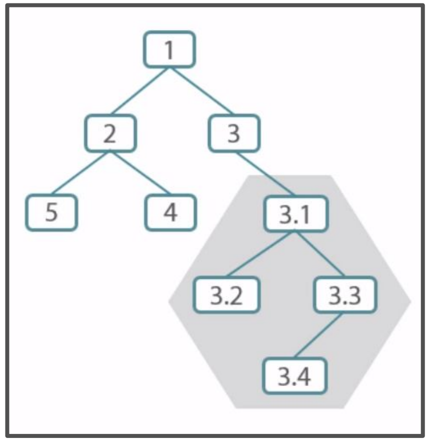
Introduction

Linux Containers (LXC)

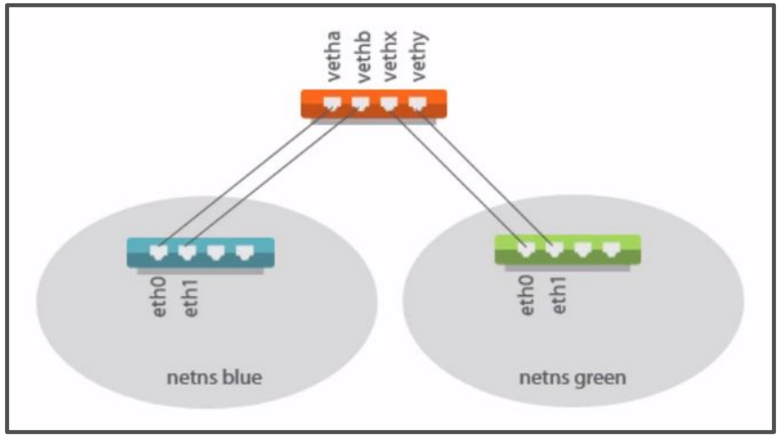
- Is an operating-system-level virtualization method
- Run multiple isolated Linux systems (containers) on a control host using a single Linux kernel.



File System



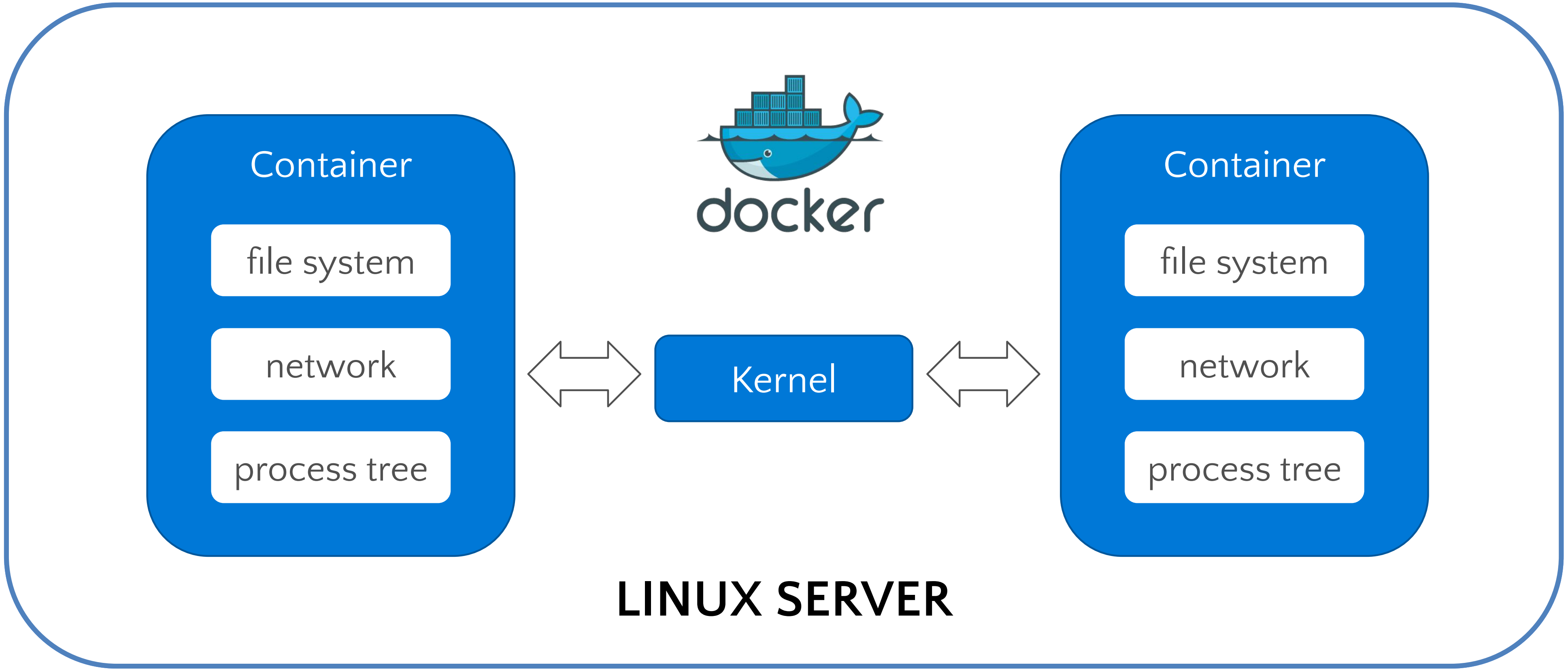
Process Tree



Networking Stacks

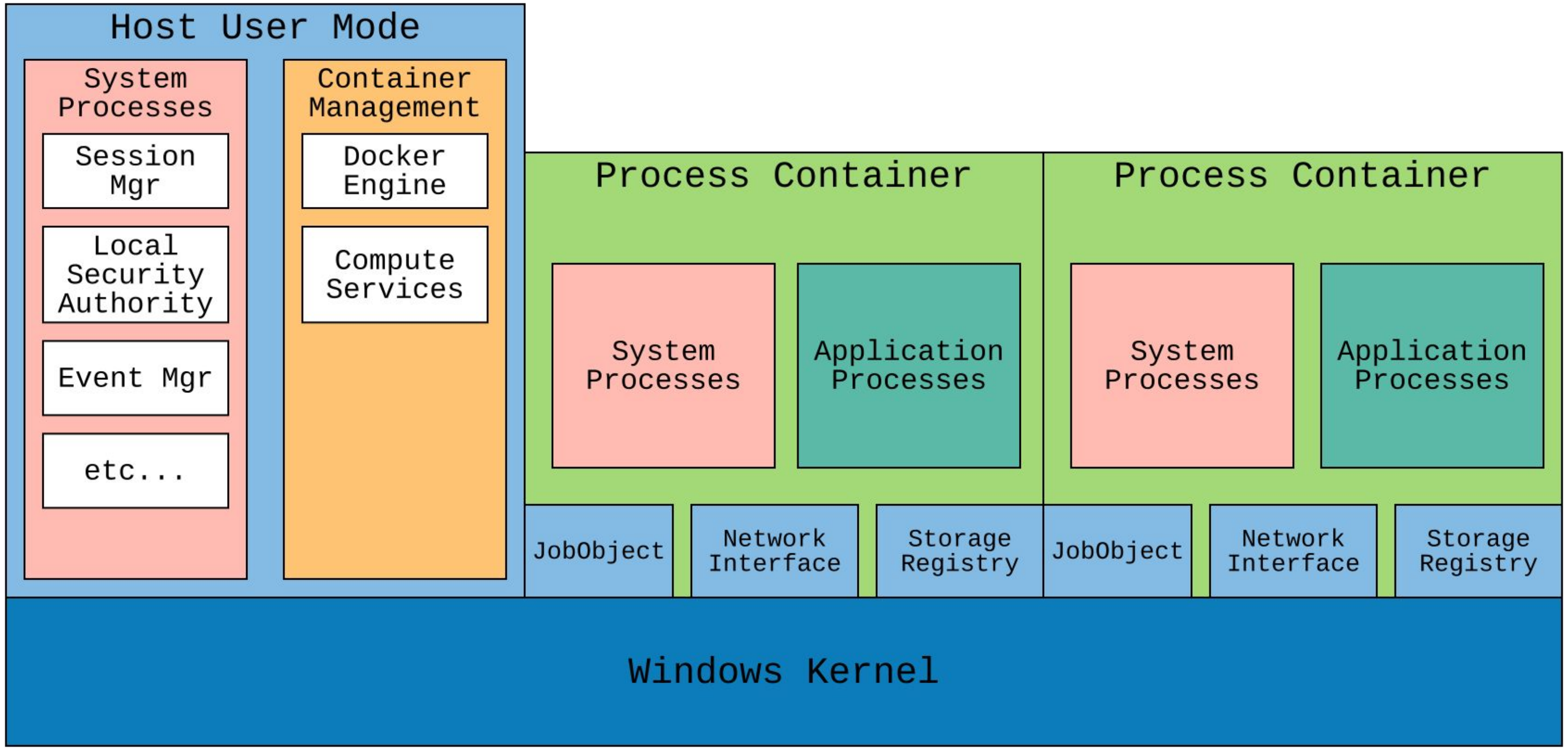
Docker Containers

- Is an actual implementation of a container technology

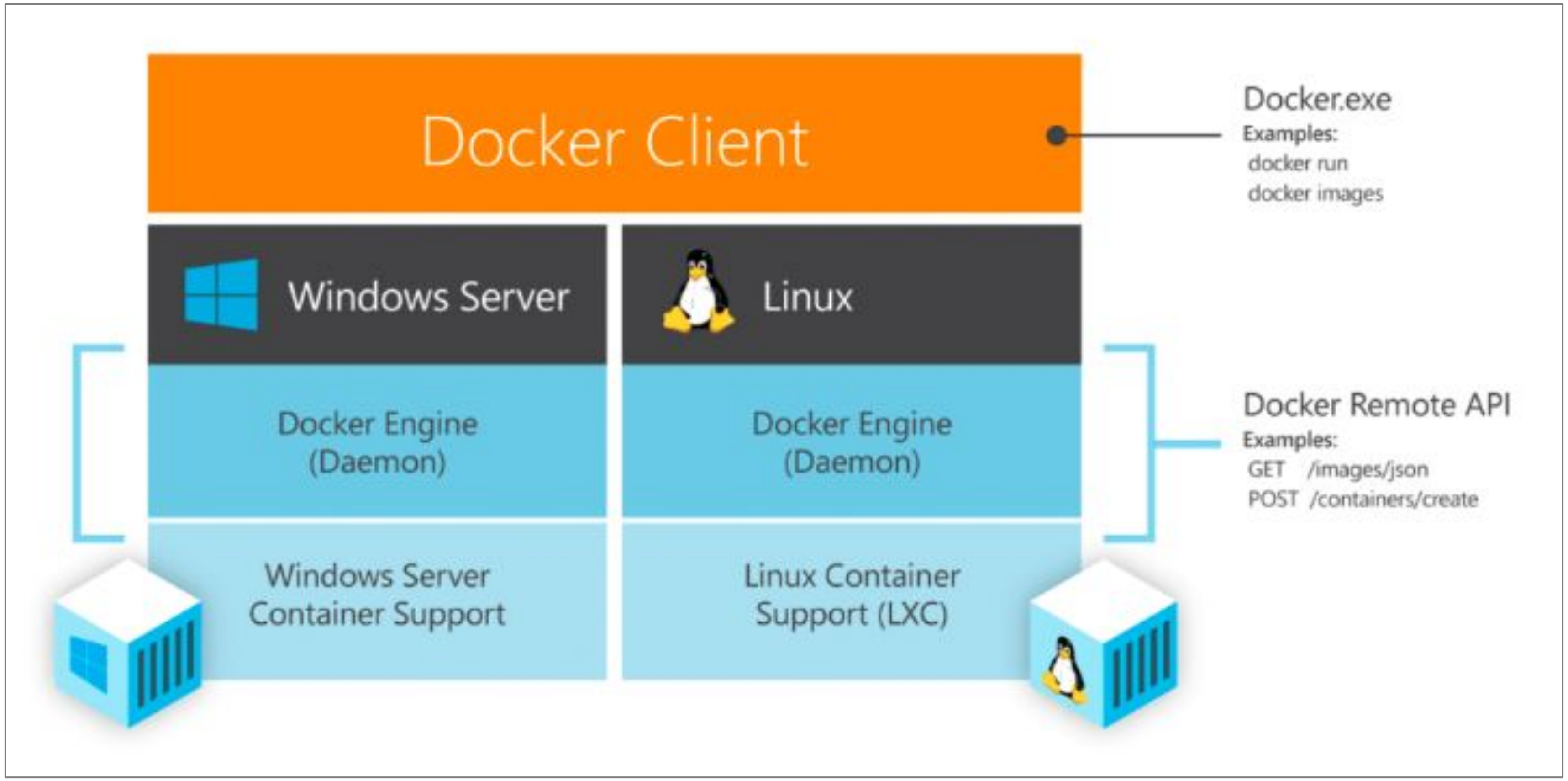


Windows Containers

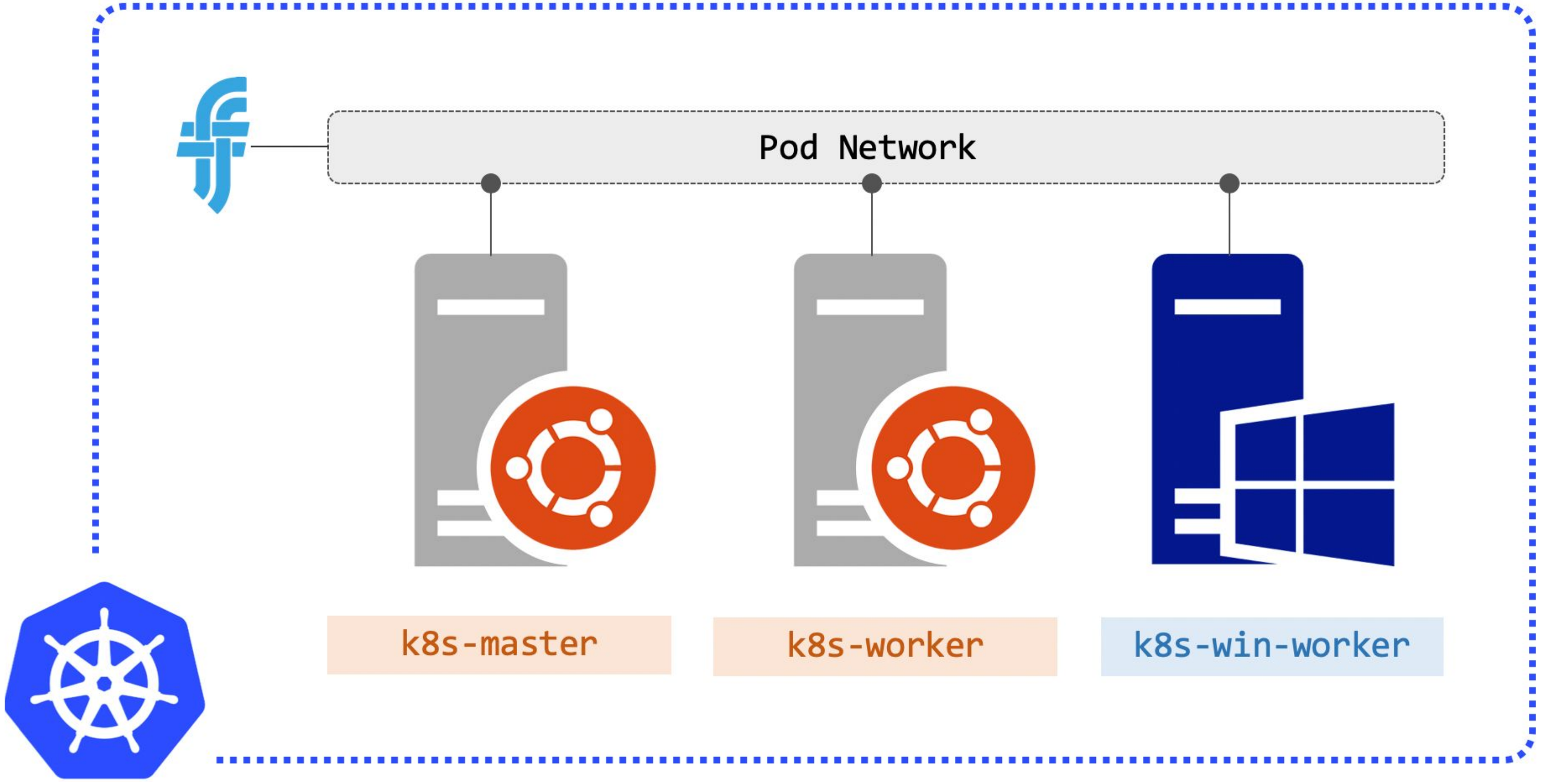
- Use “Windows Container Support” instead of LXC



Linux Containers Vs Windows Containers



Windows Containers on Kubernetes



- Pod networking is a separate component in K8s

Windows Containers on EKS (notes)

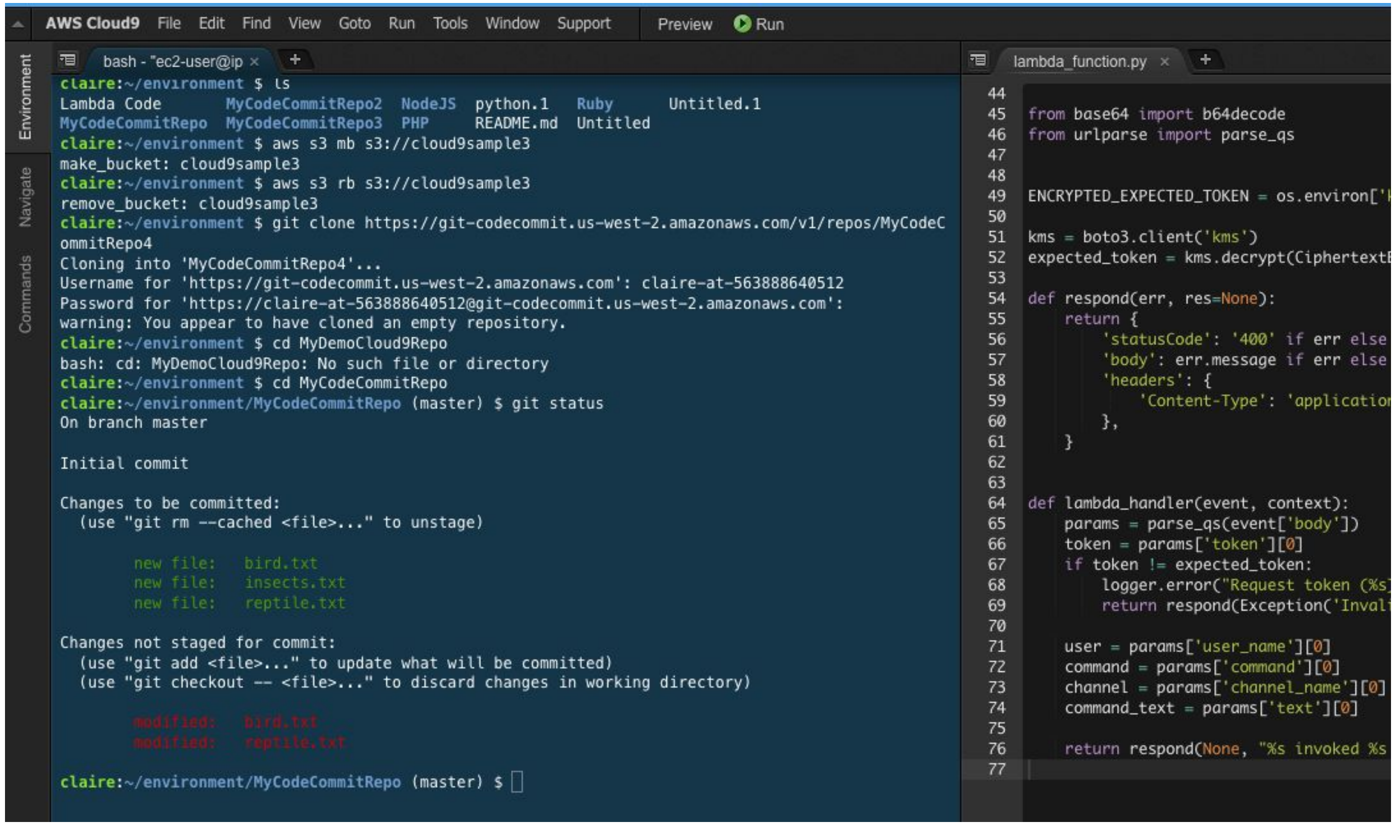
- Supported from Kubernetes version 1.14 or later
- EKS clusters must contain one or more Linux worker nodes to run core system pods that only run on Linux
- The kubelet and kube-proxy event logs are redirected to the EKS Windows Event Log and are set to a 200 MB limit
- Calico network policy enforcement has not been tested with Amazon EKS Windows nodes

Windows Containers on EKS (Limitations)

- Instance types not supported: C3, C4, D2, I2, R3 and M4 (excluding m4.16xlarge)
- Host networking mode is not supported for Windows workloads
- Windows worker nodes support one elastic network interface per node
- Group Managed Service Accounts for Windows pods and containers is a Kubernetes 1.14 alpha feature that is not supported by Amazon EKS

AWS Cloud9

- AWS Cloud9 is a cloud-based IDE that lets you write, run, and debug your code with just a browser



```
claire:~/environment $ ls
Lambda Code       MyCodeCommitRepo2  NodeJS  python.1   Ruby      Untitled.1
MyCodeCommitRepo  MyCodeCommitRepo3  PHP     README.md  Untitled
claire:~/environment $ aws s3 mb s3://cloud9sample3
make_bucket: cloud9sample3
claire:~/environment $ aws s3 rb s3://cloud9sample3
remove_bucket: cloud9sample3
claire:~/environment $ git clone https://git-codecommit.us-west-2.amazonaws.com/v1/repos/MyCodeC
ommitRepo4
Cloning into 'MyCodeCommitRepo4'...
Username for 'https://git-codecommit.us-west-2.amazonaws.com': claire-at-563888640512
Password for 'https://claire-at-563888640512@git-codecommit.us-west-2.amazonaws.com':
warning: You appear to have cloned an empty repository.
claire:~/environment $ cd MyDemoCloud9Repo
bash: cd: MyDemoCloud9Repo: No such file or directory
claire:~/environment $ cd MyCodeCommitRepo
claire:~/environment/MyCodeCommitRepo (master) $ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   bird.txt
        new file:   insects.txt
        new file:   reptile.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   bird.txt
        modified:   reptile.txt

claire:~/environment/MyCodeCommitRepo (master) $ 
```

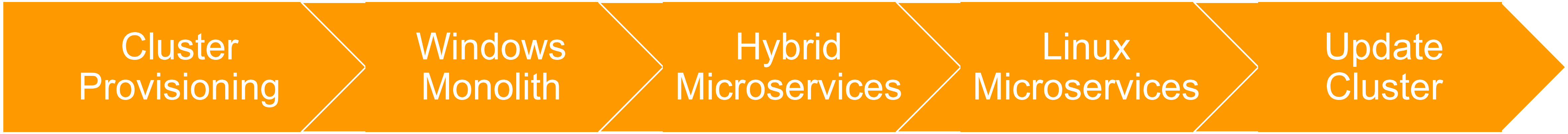
```python

from base64 import b64decode
from urlparse import parse_qs


ENCRYPTED_EXPECTED_TOKEN = os.environ['

kms = boto3.client('kms')
expected_token = kms.decrypt(CiphertextB

def respond(err, res=None):
    return {
        'statusCode': '400' if err else
        'body': err.message if err else
        'headers': {
            'Content-Type': 'applicatio
        },
    }


def lambda_handler(event, context):
    params = parse_qs(event['body'])
    token = params['token'][0]
    if token != expected_token:
        logger.error("Request token (%s
        return respond(Exception('Invali

    user = params['user_name'][0]
    command = params['command'][0]
    channel = params['channel_name'][0]
    command_text = params['text'][0]

    return respond(None, "%s invoked %s

```

Case of Study

Case of Study

Convert a windows monolith into a linux microservices

(running on kubernetes from day 1)



Cluster Provisioning

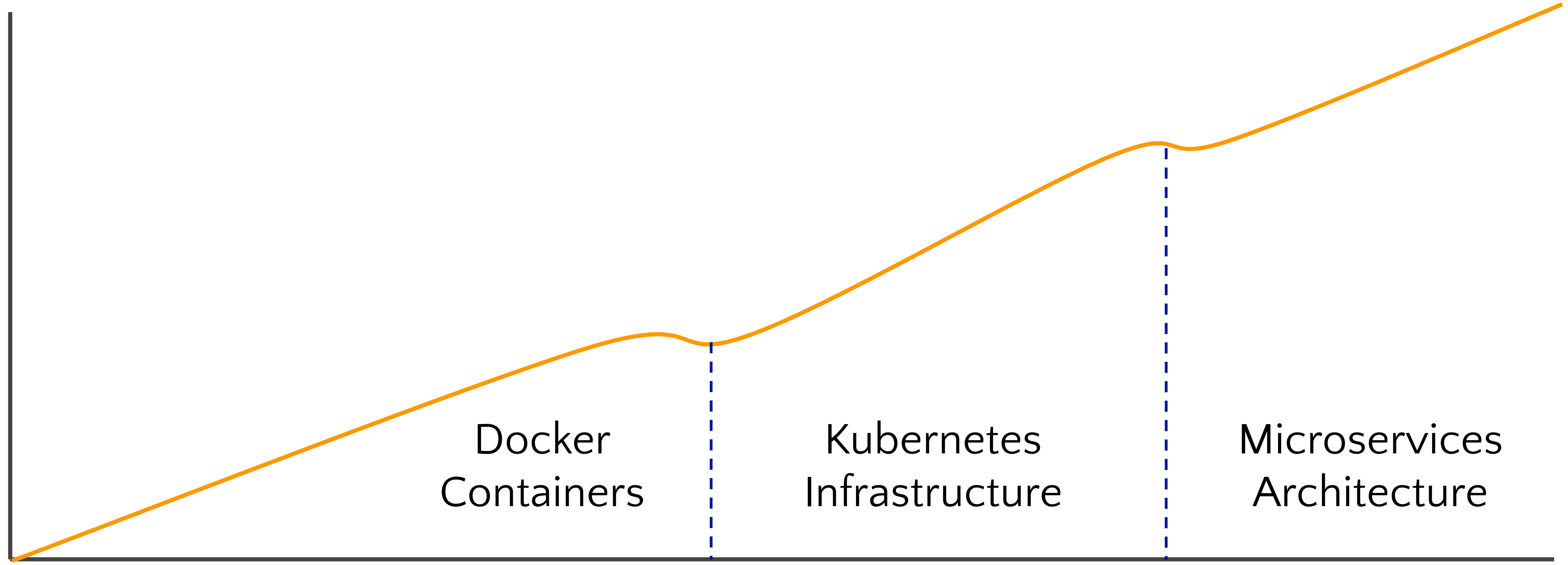
Windows Monolith

Hybrid Microservices

Linux Microservices

Update Cluster

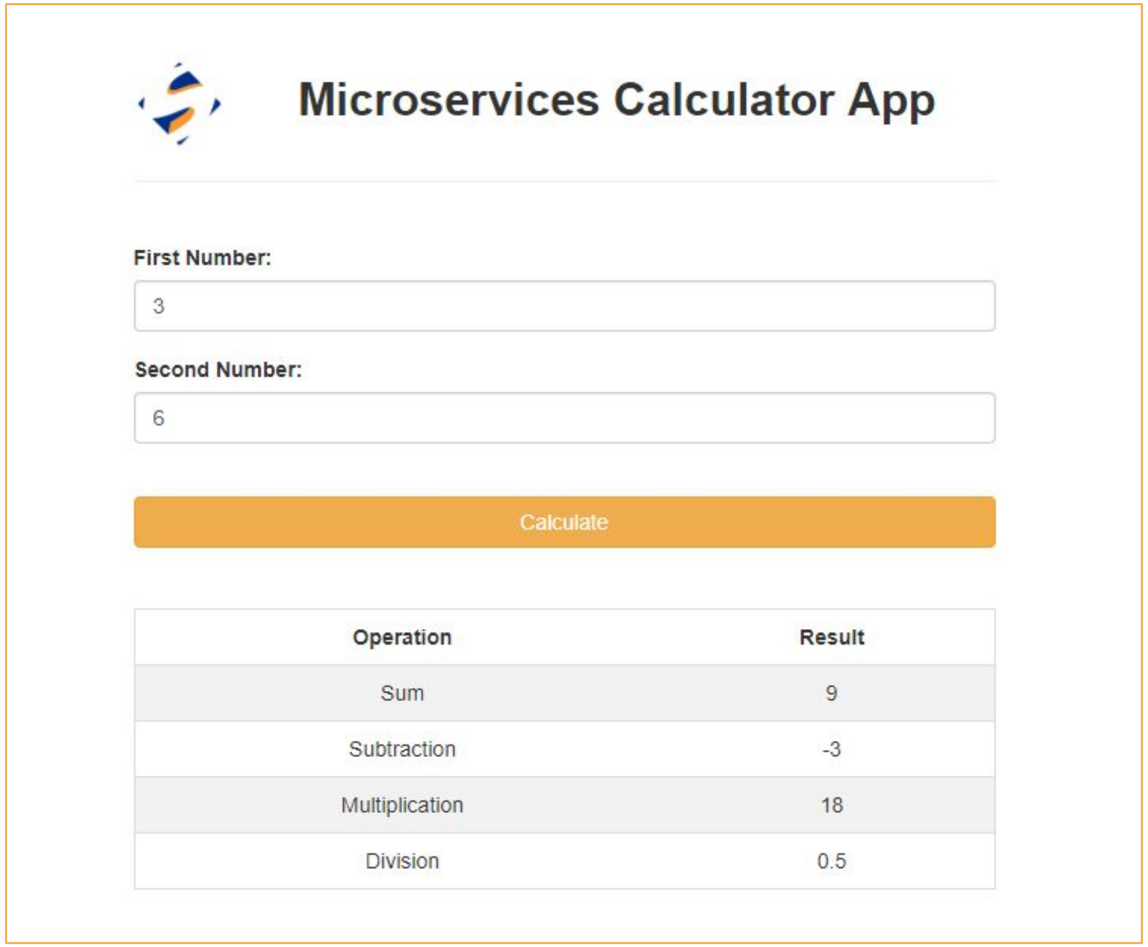
Learning Curve



Demo Application

Calculator Application

- Sum Service
- Subtraction Service
- Multiplication Service
- Division Service
- UI Service



Microservices Calculator App

First Number:

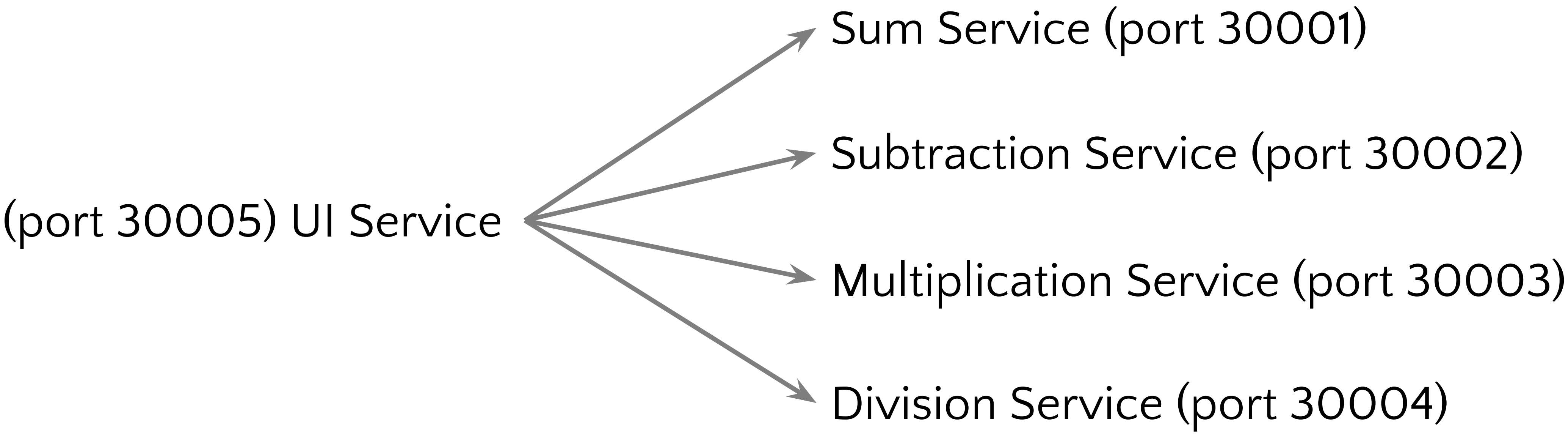
3

Second Number:

6

Operation	Result
Sum	9
Subtraction	-3
Multiplication	18
Division	0.5

Demo Application



(port 30005) UI Service

Sum Service (port 30001)

Subtraction Service (port 30002)

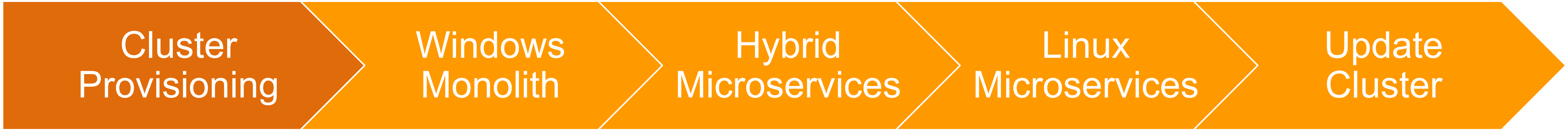
Multiplication Service (port 30003)

Division Service (port 30004)

(All services are exposed in port 3000 internally)

Let's get to work!

Implementation



Cluster Provisioning

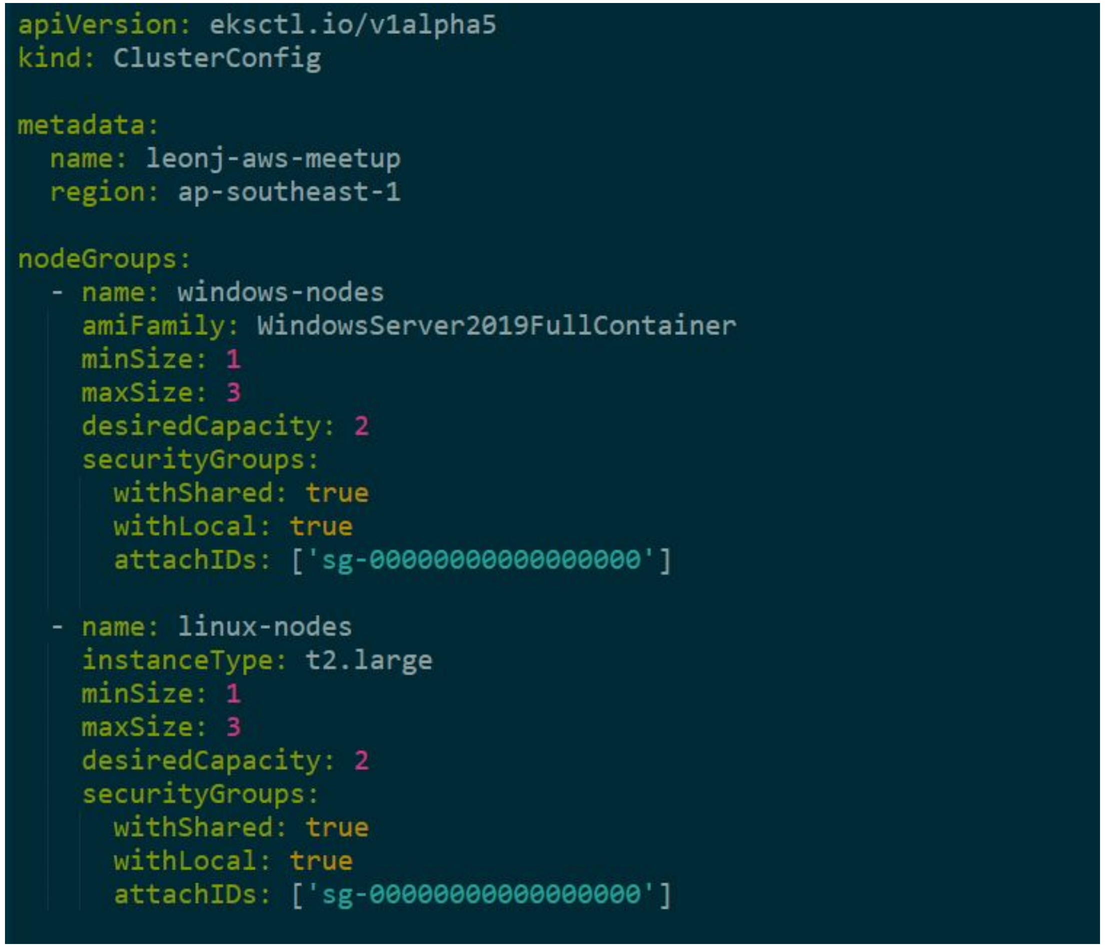
Windows Monolith

Hybrid Microservices

Linux Microservices

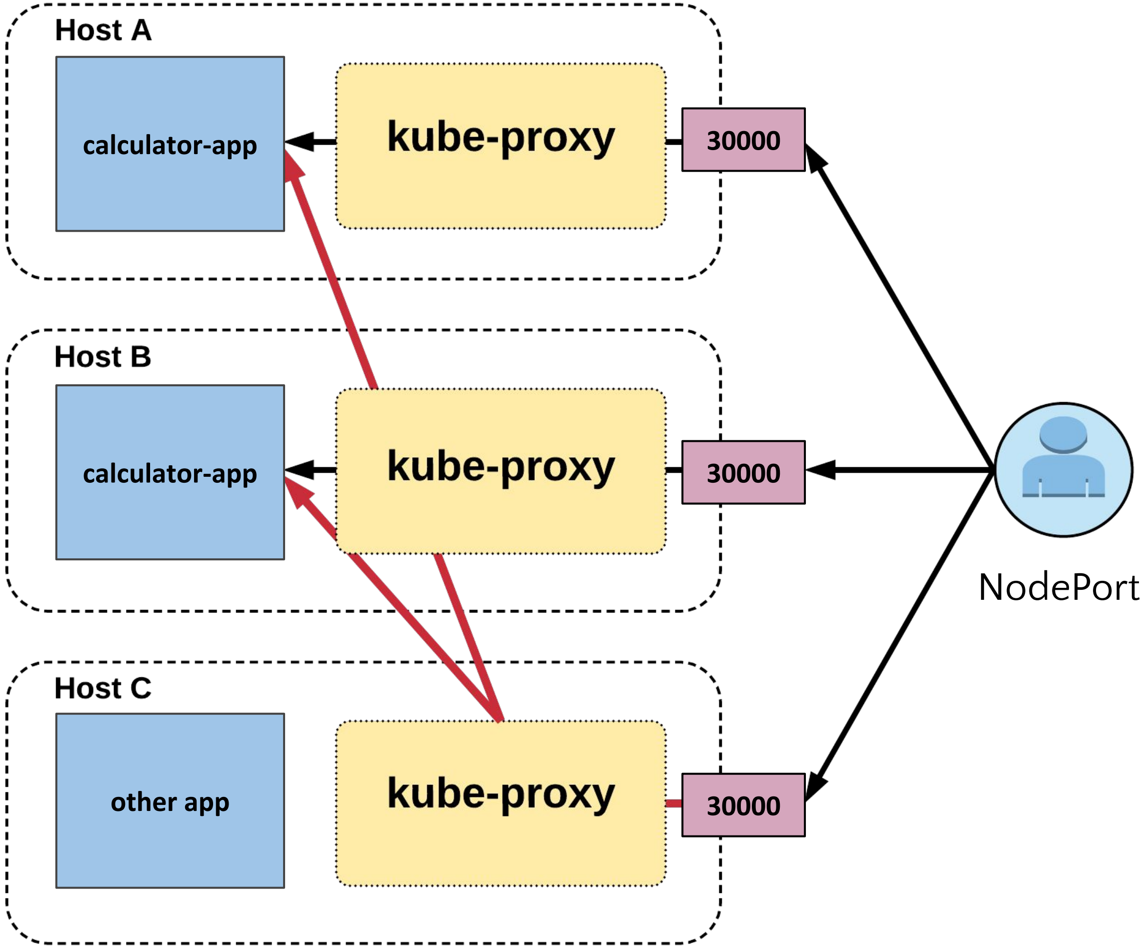
Update Cluster

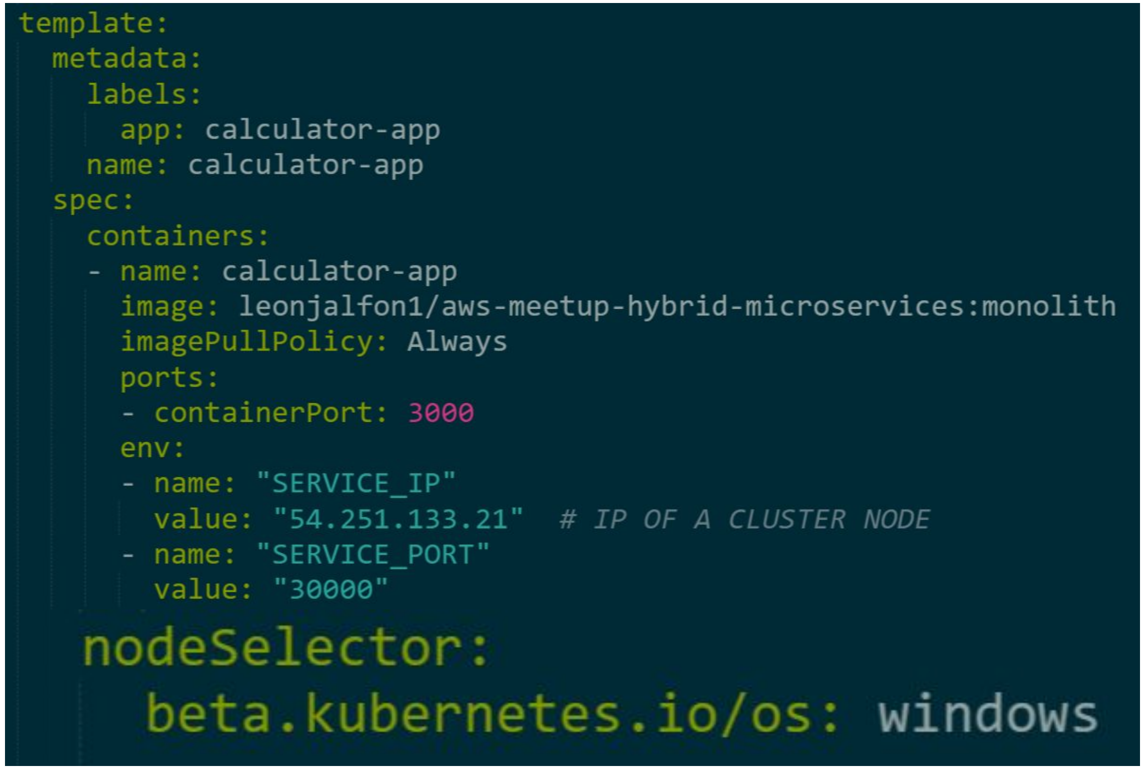
`eksctl create cluster`



```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: leonj-aws-meetup
  region: ap-southeast-1

nodeGroups:
  - name: windows-nodes
    amiFamily: WindowsServer2019FullContainer
    minSize: 1
    maxSize: 3
    desiredCapacity: 2
    securityGroups:
      withShared: true
      withLocal: true
      attachIDs: ['sg-00000000000000000']

  - name: linux-nodes
    instanceType: t2.large
    minSize: 1
    maxSize: 3
    desiredCapacity: 2
    securityGroups:
      withShared: true
      withLocal: true
      attachIDs: ['sg-00000000000000000']
```

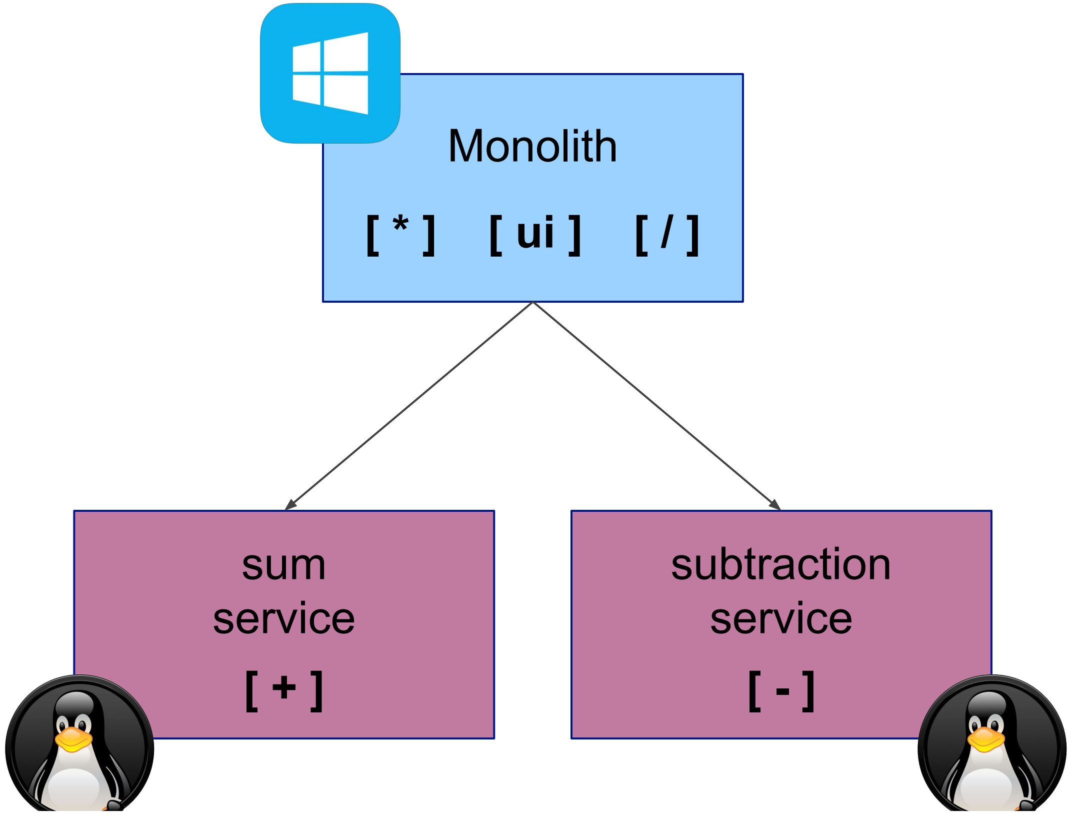
Cluster Provisioning → Windows Monolith → Hybrid Microservices → Linux Microservices → Update Cluster

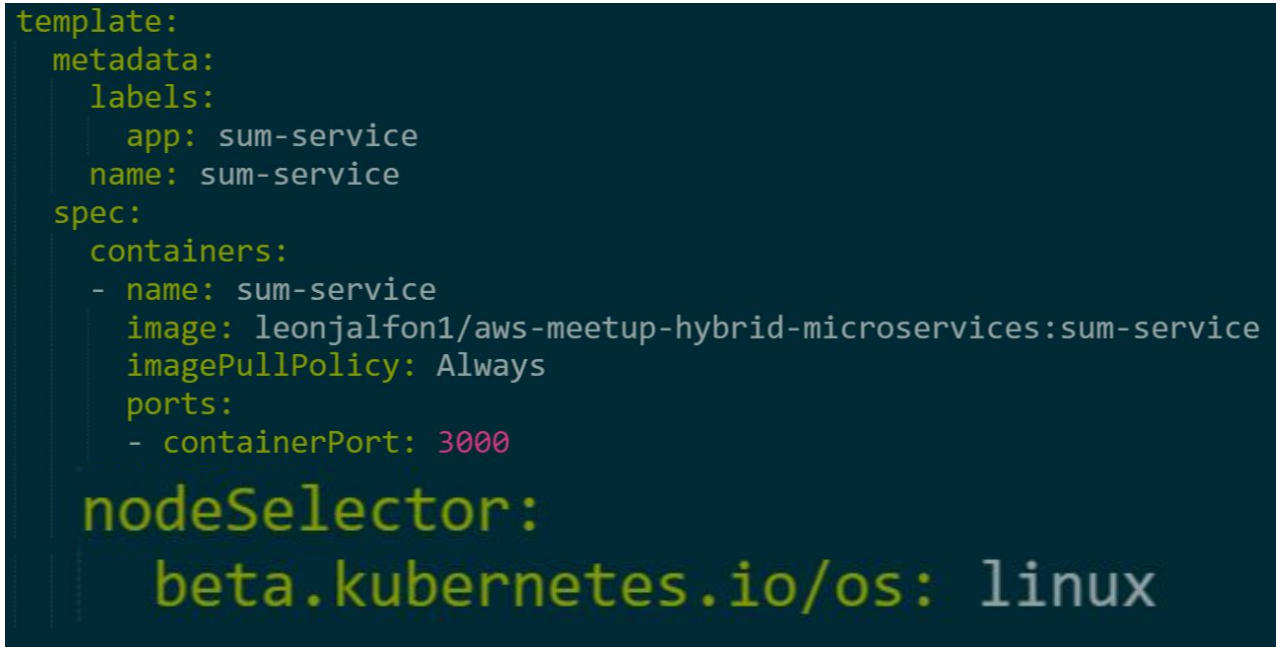




```
template:
  metadata:
    labels:
      app: calculator-app
    name: calculator-app
  spec:
    containers:
    - name: calculator-app
      image: leonjalfon1/aws-meetup-hybrid-microservices:monolith
      imagePullPolicy: Always
      ports:
      - containerPort: 3000
      env:
      - name: "SERVICE_IP"
        value: "54.251.133.21"   # IP OF A CLUSTER NODE
      - name: "SERVICE_PORT"
        value: "30000"
    nodeSelector:
      beta.kubernetes.io/os: windows
```

Cluster Provisioning → Windows Monolith → Hybrid Microservices → Linux Microservices → Update Cluster





```yaml
template:
  metadata:
    labels:
      app: sum-service
    name: sum-service
  spec:
    containers:
    - name: sum-service
      image: leonjalfon1/aws-meetup-hybrid-microservices:sum-service
      imagePullPolicy: Always
      ports:
      - containerPort: 3000
    nodeSelector:
      beta.kubernetes.io/os: linux
```

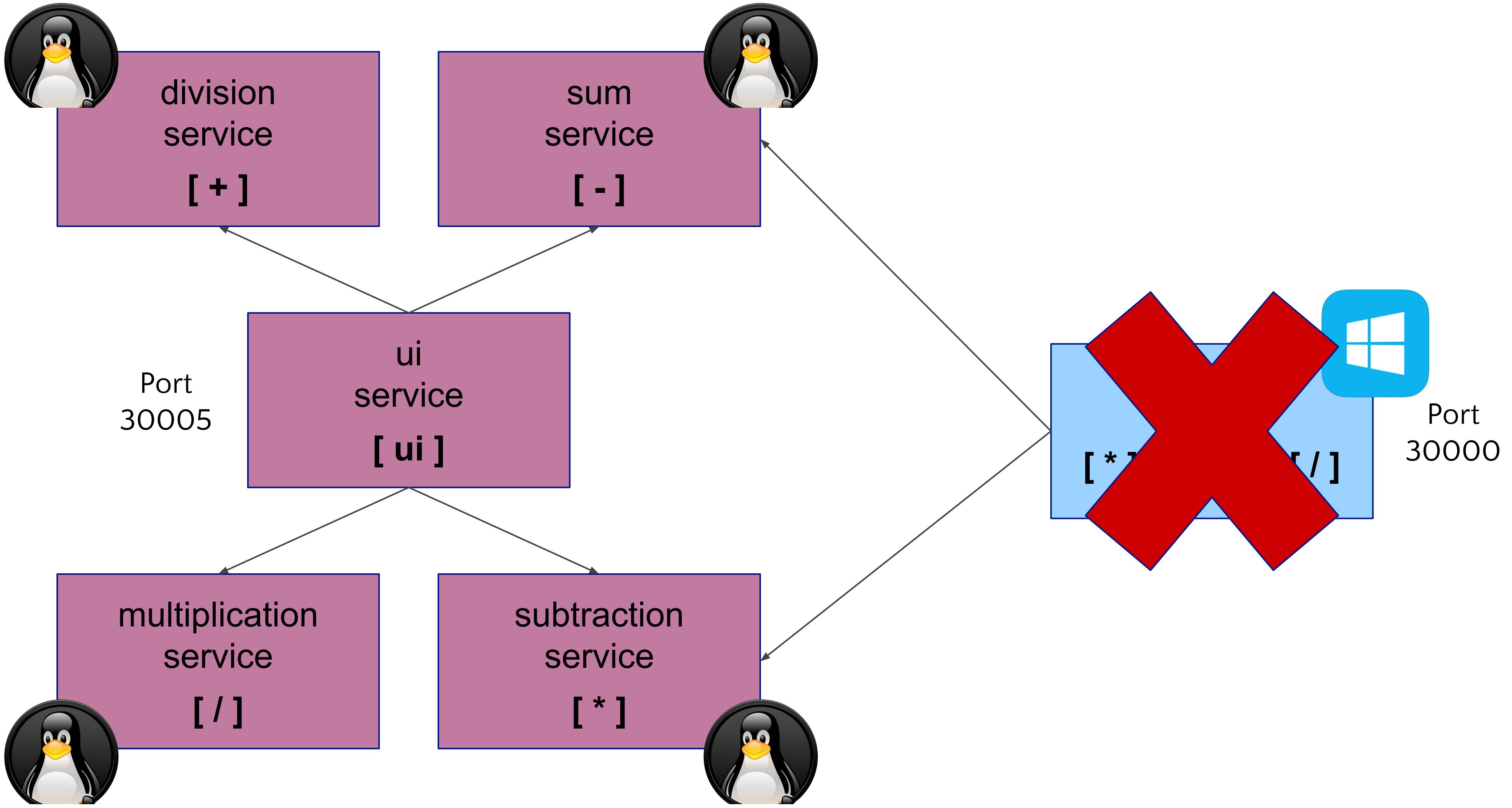
Cluster Provisioning

Windows Monolith

Hybrid Microservices

Linux Microservices

Update Cluster



division service

[+]

sum service

[-]

Port 30005

ui service

[ui]

multiplication service

[/]

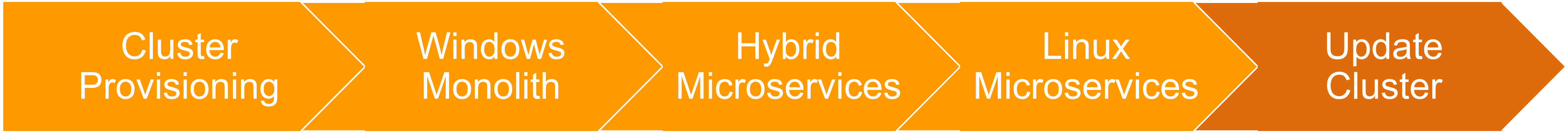
subtraction service

[*]

[*]

[/]

Port 30000



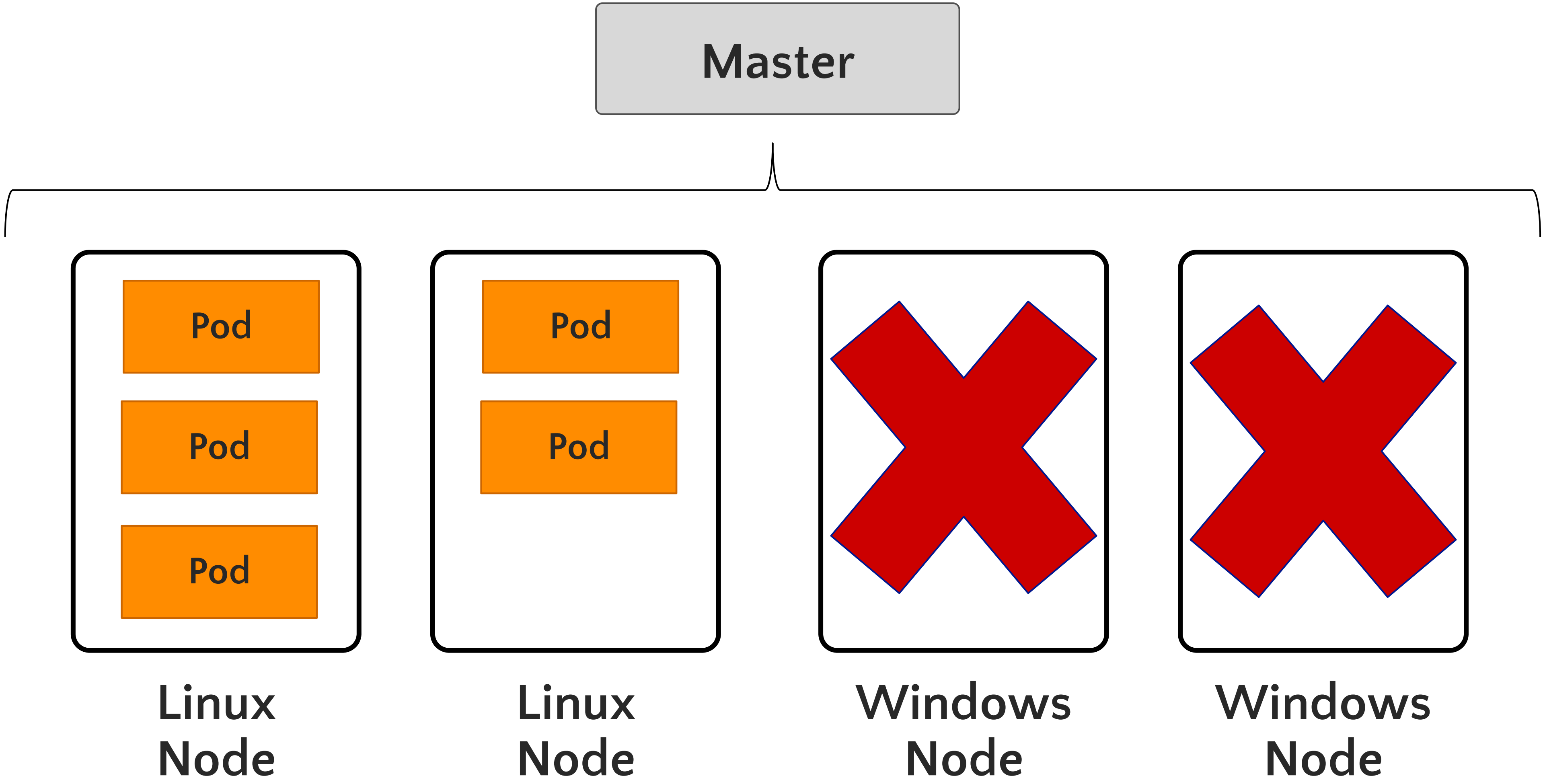
Cluster Provisioning

Windows Monolith

Hybrid Microservices

Linux Microservices

Update Cluster



Master

Pod

Pod

Pod

Pod

Pod

Linux Node

Linux Node

Windows Node

Windows Node

Summary

Summary

- Differences between windows and linux containers
- EKS windows container support considerations
- Create and manage an hybrid EKS cluster using eksctl
- Monolith to Microservices transformation
- Manage windows and linux workloads in kubernetes

Questions

Thank you!

Leon Jalfon

https://github.com/leonjalfon1/aws-meetup-hybrid-eks