VM BASED TRUSTED EXECUTION ENVIRONMENTS ON KUBERNETES

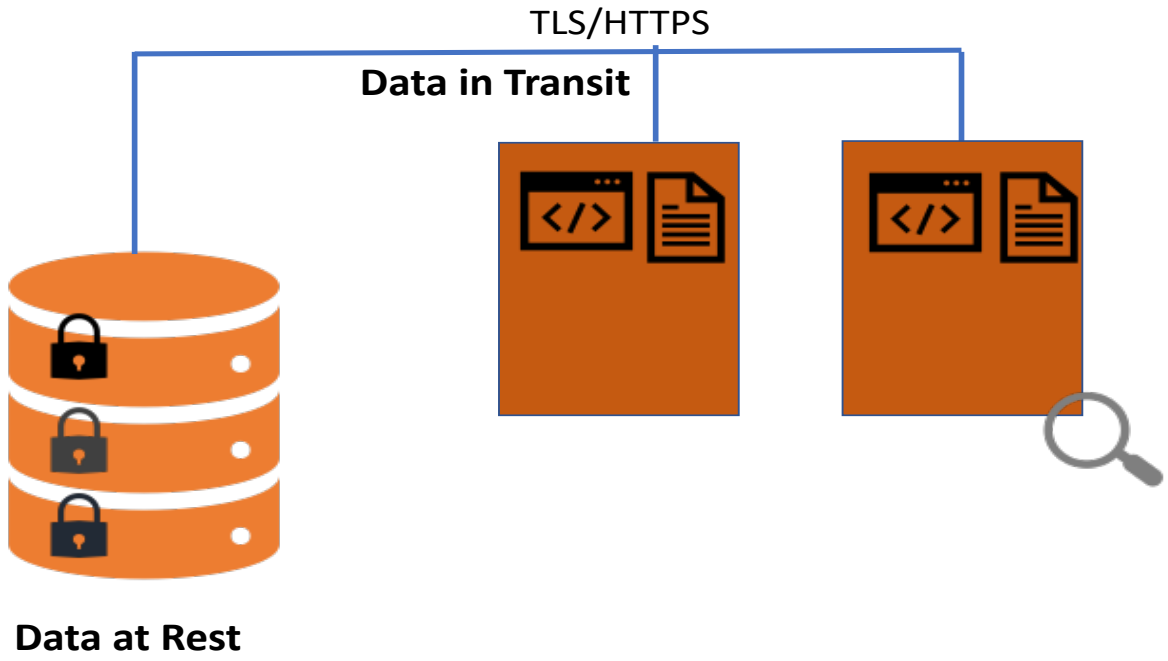
Project **Raksh** *(Protect)*

https://github.com/ibm/raksh

AGENDA

- Problem
- Trusted Execution Environment
- Raksh Overview
- Next Steps

PROBLEM – PROTECT DATA IN-USE



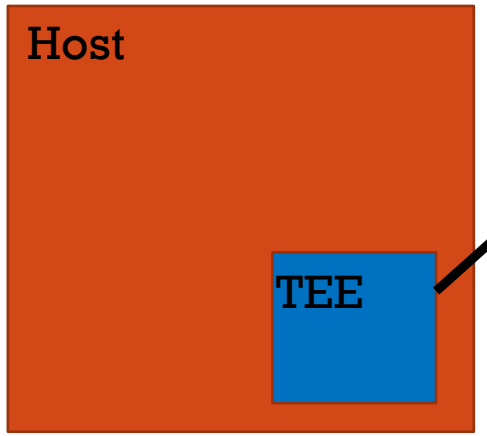
Data in Use ??

- From other software
- Malicious Admins
- Compromised host/hypervisor

How do you maintain data confidentiality when running sensitive workloads on your premises or in the cloud ?

SOLUTION - TRUSTED EXECUTION ENVIRONMENT (TEE)

- TEE is a protected area within the host used for execution of sensitive workloads
- TEEs provide confidentiality guarantees for code and data running within them. IOW the running workload can't be seen from outside the TEE.



Types

- **Process based**
  - Process address space obfuscated from the OS
  - Examples - Intel SGX
  - Application code modification required
  - No virtualization overheads.
- **VM based**
  - VM address space obfuscated from the hypervisor
  - Examples – Intel TME/MKTME, AMD SEV, IBM Power9 PEF
  - No modifications to application code required
  - Virtualization overhead

How to use TEEs with Kubernetes (Openshift, IKS, AKS, PKS …)?

PROJECT RAKSH

INTRODUCTION

- **Salient Points**
  - **Raksh** - means Protect (*https://github.com/ibm/raksh*)
  - Opensource project to enable deployment of **VM based TEEs** in a Kubernetes cluster
  - Introduces **Secure Containers** resource in K8s which are **containers protected by VM based TEEs**
  - Introduces **encryption for K8s application spec** (pod.yaml, deployment.yaml etc)
  - No changes to K8s application deployment workflow
  - Leverages Kata VM container runtime
  - Built on Kubernetes Operator pattern
  - CLI tool (*rakshctl*)

- **Dependencies**
  - Supported Hardware
  - Kata container runtime

RAKSH DETAILS

SECURE CONTAINERS

- **Raksh** defines **Secure container** as a container that runs in VM based TEE. The container is isolated and protected using VM based TEE.
- **Secure Container** in **Raksh** is implemented using the **Operator** pattern.
- Following are the Custom Resource Definitions (CRDs)
  - **SecureContainer** - Manages the lifecycle of the Secure Container
  - **SecureContainerImage -** Manages the lifecycle of VM image used to execute the Secure Container
  - **SecureContainerImageConfig -** Manages the configuration (download location etc) of the SecureContainerImage

SECURE CONTAINER CRDS

```
apiVersion: securecontainers.k8s.io/v1alpha1
kind: SecureContainerImageConfig
metadata:
  name: <config-resource-name>
spec:
  imageDir: <host-path-to-store-image>
  runtimeClassName: kata-containers
```

SecureContainerImageConfig

- **Kata VM kernel and initrd location**
- **Kata Runtime Class to use**

```
apiVersion: securecontainers.k8s.io/v1alpha1
kind: SecureContainerImage
metadata:
  name: <image-resource-name>
spec:
  vmImage: <container-image-name>:<tag>
  imagePullSecrets:
    - name: <registry-secret>
  imagePullPolicy: Always
  SecureContainerImageConfigRef:
    name: <config-resource-name>
```

SecureContainerImage

- **Container image with Kata VM kernel and initrd**
- **Download and Extract to location specified by SecureContainerImageConfig**

```
apiVersion: securecontainers.k8s.io/v1alpha1
kind: SecureContainer
metadata:
  name: <securecontainer-resource-name>
object:
  apiVersion: v1
  kind: Pod
  metadata:
    name: <app-name>
  spec:
    containers:
      <…>
    volumes:
      <…>
spec:
  SecureContainerImageRef:
    name: < image-resource-name>
```

SecureContainer

- **Secure Container resource to deploy app container protected using VM based TEE**

CUSTOM KATA AGENT

- Handle container image lifecycle (download, extract, decryption) inside the VM
- Execute the containers inside VM

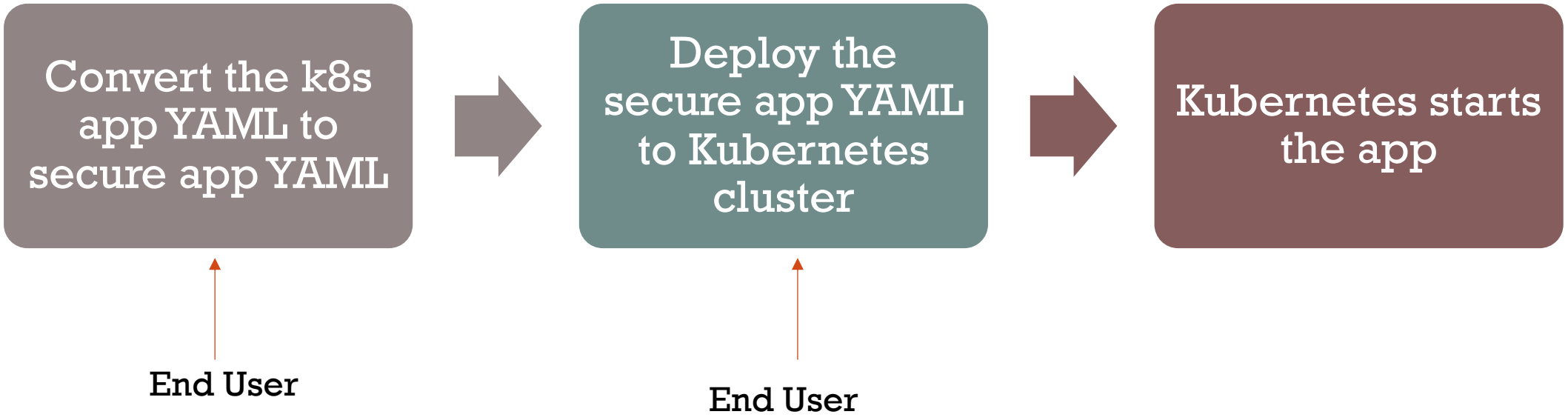
RAKSH CLI

- **rakshctl**
  - CLI to convert an existing Kubernetes application specification (pod.yaml, deployment.yaml etc) to support deployment leveraging VM based TEE

APPLICATION DEPLOYMENT WORKFLOW

- **User Story**
  - As a user I want to deploy and run my Kubernetes application securely with minimal disruption to my workflow and protect the data in-use from malicious entities

STEPS


Convert the k8s app YAML to secure app YAML
Deploy the secure app YAML to Kubernetes cluster
Kubernetes starts the app
End User
End User

GENERATE SECURE APP YAML

Input	Input App YAML (pod.yaml)
Read	Read the container spec and create Configmap from the spec
Encrypt	Encrypt the Configmap using Symmetric Key and random unique string (*nonce*)
Create Initrd	Store the Symmetric Key, Nonce in the Kata Initrd
Create Container Image	Create container image consisting of Kata kernel and Initrd. *This will be the SecureContainerImage resource*
Output	Output modified App YAML *This will be the SecureContainer resource*

GENERATE SECURE APP YAML

Convert the YAML to secure YAML

rakshctl image create -i nginx-securecontainerimage --initrd kata-containers-initrd.img --symmKey /home/pradipta/symm_key --vmlinux vmlinuz --scratch sc-scratch:latest --filename /home/pradipta/nginx.yaml nginx-securecontainerimage

1. Create a ConfigMap of the container spec

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: nginx
  namespace: default
  name: nginx
spec:
  containers:
    - image: nginx:latest
      imagePullPolicy: IfNotPresent
      name: nginx
      ports:
      - containerPort: 80
        protocol: TCP
```

nginx.yaml

1

```
apiVersion: v1
data:
  nginx: |
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: IfNotPresent
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
kind: ConfigMap
metadata:
  name: configmap-nginx
```

ConfigMap

CONT...

Convert the YAML to secure YAML

rakshctl image create -i nginx-securecontainerimage --initrd kata-containers-initrd.img --symmKey /home/pradipta/symm_key --vmlinux vmlinuz --scratch sc-scratch:latest --filename /home/pradipta/nginx.yaml nginx-securecontainerimage

1. *Create a ConfigMap of the container spec*
2. Encrypt the ConfigMap with symmetric key

```
apiVersion: v1
data:
  nginx: |
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: IfNotPresent
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
kind: ConfigMap
metadata:
  name: configmap-nginx
```

ConfigMap

2

```
apiVersion: v1
data:
  nginx:
6qvygg8md7bXfyX3Y9cpZxUp4eZA0kKmWBirrpJv/WEGkrdLYrdtqxdqm4cGL
G4++06d2iGTaB+5SDjjDwf05T+9a2iUAdHmRngHcQNAzkKK2RCnR4Zkt0cXDa
EP+w5mbugH0xdqGm8SoX4IgvWGi2toq1CUcc8OmgTX42g0NruTZbrNv5Ncc
yS7+kR7Iib6vaMI24E=
kind: ConfigMap
metadata:
  name: secure-configmap-nginx
```

Encrypted ConfigMap

CONT...

Convert the YAML to secure YAML

rakshctl image create -i nginx-securecontainerimage --initrd kata-containers-initrd.img --symmKey /home/pradipta/symm_key --vmlinux vmlinuz --scratch sc-scratch:latest --filename /home/pradipta/nginx.yaml nginx-securecontainerimage

1. *Create a ConfigMap of the container spec*
2. *Encrypt the ConfigMap with symmetric key (key either added to lockbox or vault)*
3. Create a modified container spec to use encrypted ConfigMap (uses dummy image which just sleeps. Actual container image will be pulled inside the Kata VM)

3

```
apiVersion: v1
data:
  nginx:
6qvygg8md7bXfyX3Y9cpZxUp4eZA0kKmWBirrpJv/WEGkrdLYrdtqxdq
m4cGLG4++06d2iGTaB+5SDjjDwf05T+9a2iUAdHmRngHcQNAzkKK2R
CnR4Zkt0cXDaEP+w5mbugH0xdqGm8SoX4IgvWGi2toq1CUcc8OmgT
X42g0NruTZbrNv5NccyS7+kR7Iib6vaMI24E=
kind: ConfigMap
metadata:
   name: secure-configmap-nginx
```

```
apiVersion: securecontainers.k8s.io/v1alpha1
kind: SecureContainer
metadata:
  name: secure-nginx
object:
  apiVersion: v1
  kind: Pod
  metadata:
    labels:
      app: nginx
    name: nginx
spec:
    containers:
    - image: sc-scratch:latest
      imagePullPolicy: IfNotPresent
      name: nginx
      ports:
      - containerPort: 80
        protocol: TCP
      resources: {}
      volumeMounts:
      - mountPath: /etc/raksh
        name: secure-volume-nginx
        readOnly: true
    volumes:
    - configMap:
        items:
        - key: nginx
          path: kavach.properties
        name: secure-configmap-nginx
      name: secure-volume-nginx
spec:
  SecureContainerImageRef:
    name: nginx-securecontainerimage
```

nginx-sc.yaml

Dummy Image (sleep) running on host

Mount the encrypted ConfigMap inside the Kata VM

Reference to the encrypted ConfigMap

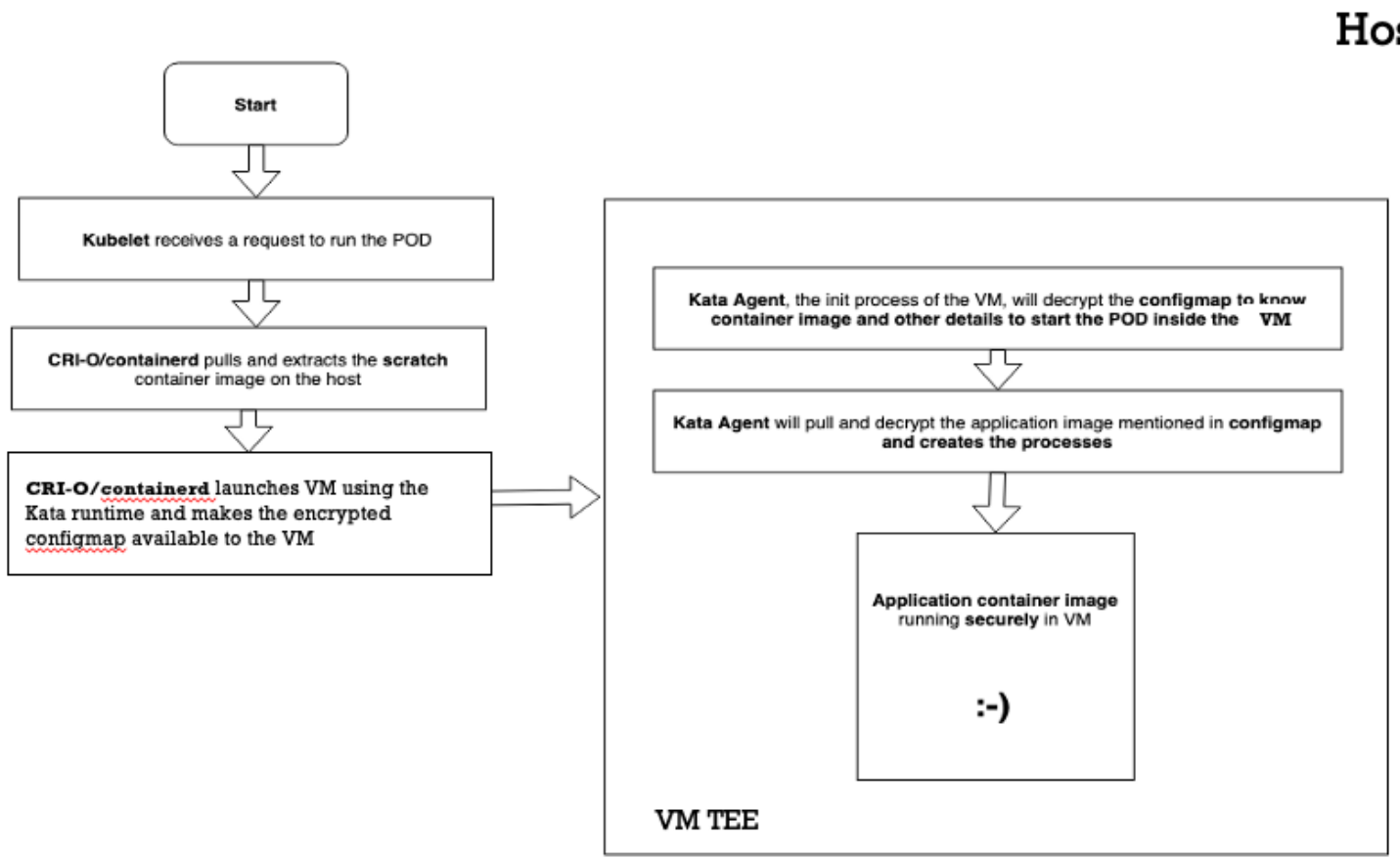
Kata VM image used

DEPLOY THE WORKLOAD

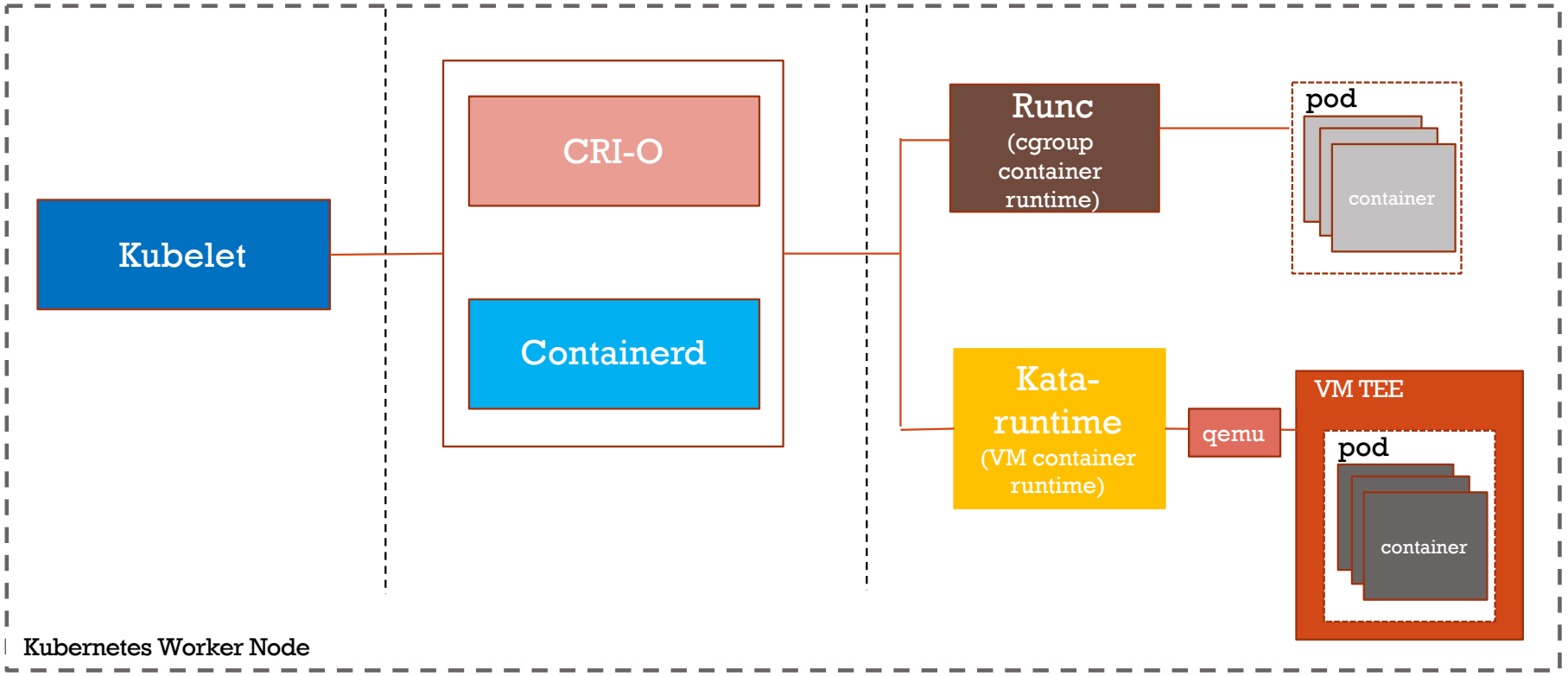
Deploy the Workload

kubectl create -f nginx-sc.yaml

- All operations w.r.to customer workload is handled inside the Kata VM which is the VM based TEE (AMD SEV, IBM PEF etc)
- The secure workload spec is immutable and it's not possible for anyone with admin privileges to the Kubernetes cluster to modify the workload spec.



COMPONENTS



VM TEE – AMD SEV, IBM PEF, INTEL MKTME

DEMO

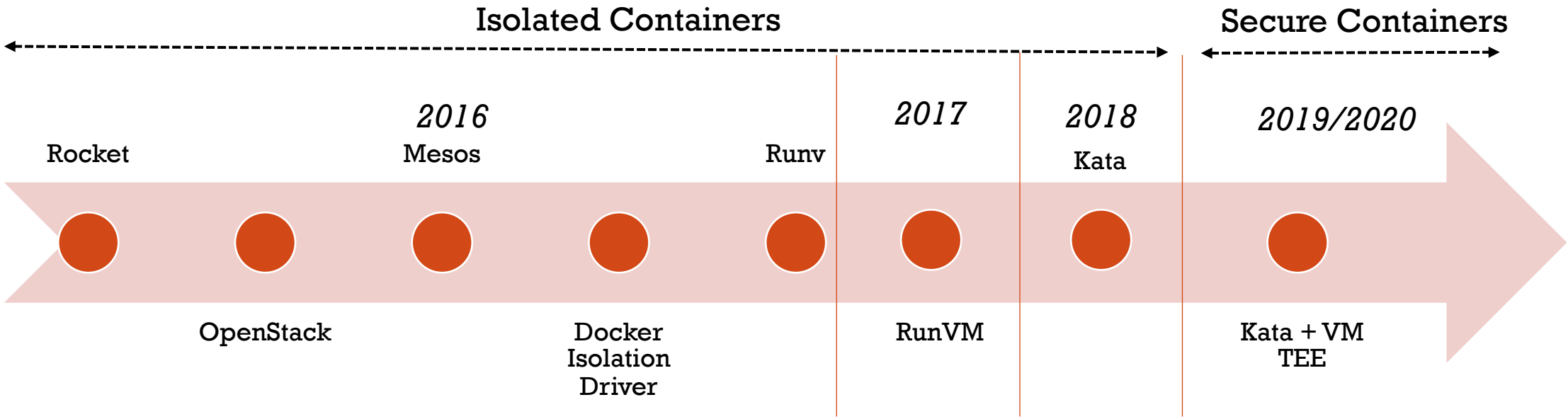
1. Create VM image (Kata initrd and vmlinux)
2. Create SecureContainerImageConfig CRD
    1. VM image name, install location etc.
3. Create SecureContainerImage CRD
    1. Docker image with VM kernel, initrd
4. Show vault details stored as Kubernetes secret
5. Show regular app yaml and secure app yaml
6. Deploy the secure app yaml

Demo Link - https://asciinema.org/a/290233

REFERENCES

- Introducing Project Raksh - http://bit.ly/project_raksh
- What You Define is What You Deploy - http://bit.ly/wydiwud
- Raksh Github Link - https://github.com/ibm/raksh

JOURNEY TO SECURE CONTAINERS



References

- http://bit.ly/isolatedcontainers
- https://github.com/harche/runvm
- https://github.com/moby/moby/issues/29454
- https://katacontainers.io/posts/kata-containers-ibm-power/
- https://github.com/opencontainers/image-spec/issues/747
- https://github.com/kata-containers/runtime/issues/61
- https://docs.google.com/document/d/1NjM58FZFRLtK2qh90WYnk-1nd8ili1d433d8pGwri6g/edit#heading=h.pext71slr0u9
- https://docs.google.com/document/d/1zSk397hmaRuN6LkXzjYjltjkxG0SZRw5pr4ZYzfxC4c/edit#heading=h.nupw5ka6qe7w
- https://github.com/ibm/raksh