

# RSAC Conference™ 2023

San Francisco | April 24 – 27 | Moscone Center

SESSION ID: CSCO-R05

## Lateral Movements in Kubernetes



#RSAC

**Yossi Weizman**

Senior Security Researcher  
Microsoft

<https://t.me/learningnets>

# Disclaimer



Presentations are intended for educational purposes only and do not replace independent professional judgment. Statements of fact and opinions expressed are those of the presenters individually and, unless expressly stated to the contrary, are not the opinion or position of RSA Conference™ or any other co-sponsors. RSA Conference does not endorse or approve, and assumes no responsibility for, the content, accuracy or completeness of the information presented.

Attendees should note that sessions may be audio- or video-recorded and may be published in various media, including print, audio and video formats without further notice. The presentation template and any media capture are subject to copyright protection.

© 2023 RSA Conference LLC or its affiliates. The RSA Conference logo and other trademarks are proprietary. All rights reserved.

# Agenda

- Kubernetes – Overview & identities
- Inner-cluster lateral movement
- Cluster-to-cloud lateral movement
- Cross-cloud lateral movement
- Mitigations & Detections
- Key takeaways



**RSA**Conference™2023

**Stronger  
Together**



# Kubernetes: Overview & identities

# Kubernetes – overview



**Containers** - A unit of software that packages the code and all its dependencies. The executable package is called **image**. At runtime, containers run as **isolated** software in the host OS. Containers run by a container engine such as Docker or ContainerD

**Kubernetes** - container-orchestration system. Basically, manages a cluster of computers (nodes), each one is running a Container engine. Enables to manage and scale containerized environments.

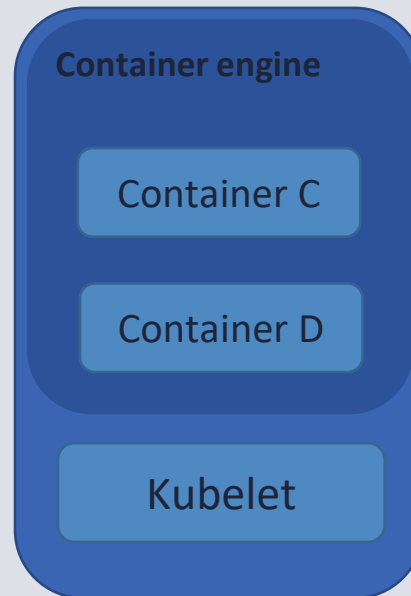
## K8s control plane



### Node 1



### Node 2



### Node 3



## K8s control plane

In managed clusters:  
Fully managed  
by the cloud provider

API Server

scheduler

etcd

Controller manager

### Node 1

Container engine

Container A

Container B

Kubelet

### Node 2

Container engine

Container C

Container D

Kubelet

### Node 3

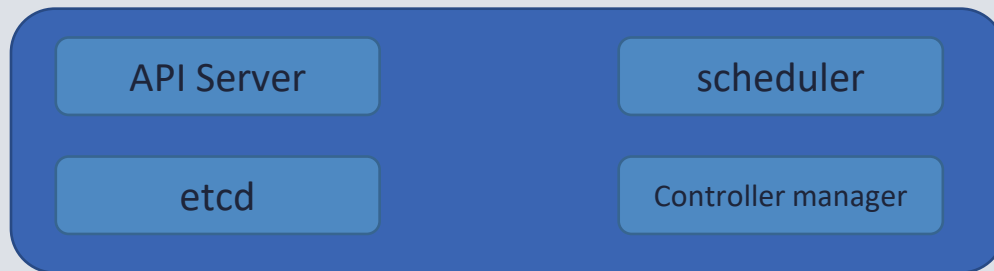
Container engine

Container E

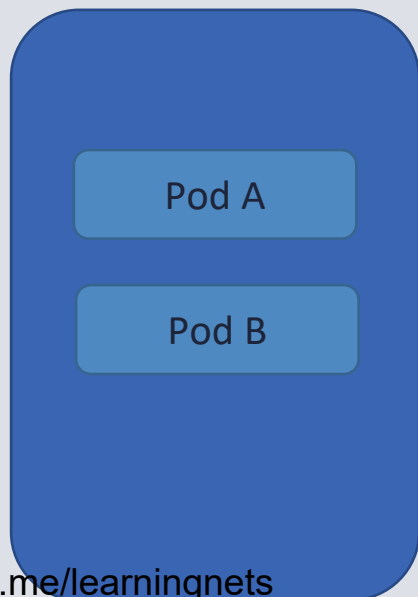
Container F

Kubelet

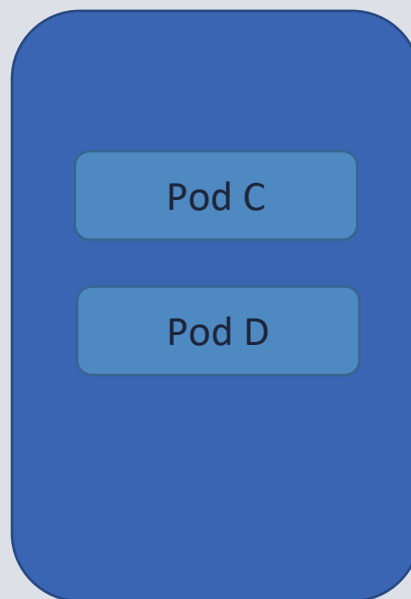
## K8s control plane



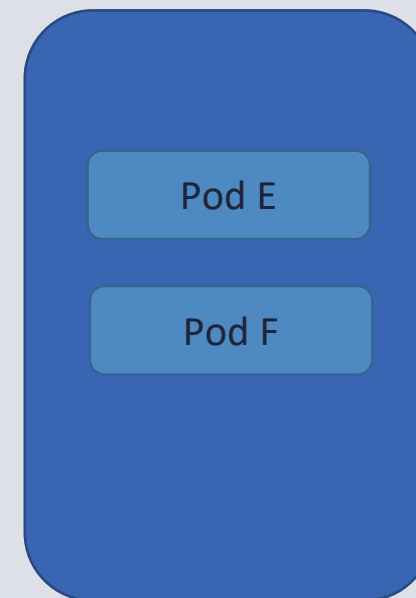
### Node 1



### Node 2



### Node 3



# Identity types in Kubernetes

## Three main areas:

- How users (or applications) from outside the cluster authenticate with the cluster.
- How workloads in the cluster authenticate within the cluster.
- How workloads in the cluster authenticate with resources in the cloud outside the cluster.

# Identity types in Kubernetes



## Three main areas:

- How users (or applications) from outside the cluster authenticate with the cluster.
- **How workloads in the cluster authenticate within the cluster.**
- **How workloads in the cluster authenticate with resources in the cloud outside the cluster.**

# Kubernetes AuthN & AuthZ

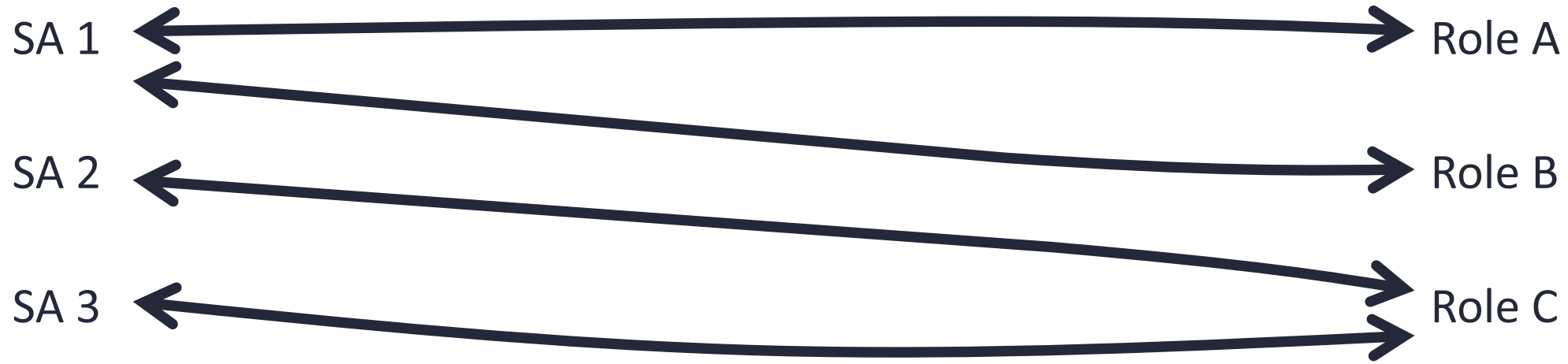


- Service account = Identity of an application in Kubernetes
- RBAC – Role-based access control
- Roles = set of rules (permissions)
- Role binding = assignment of a role to a user\service account

# Kubernetes AuthN & AuthZ

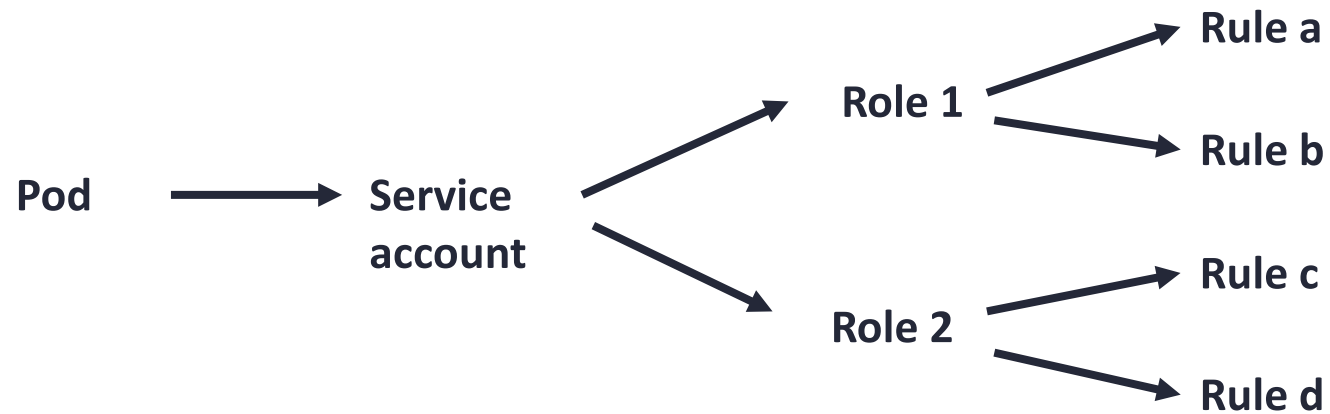


- Service account = Identity of an application in Kubernetes
- RBAC – Role-based access control
- Roles = set of rules (permissions)
- Role binding = assignment of a role to a user\service account

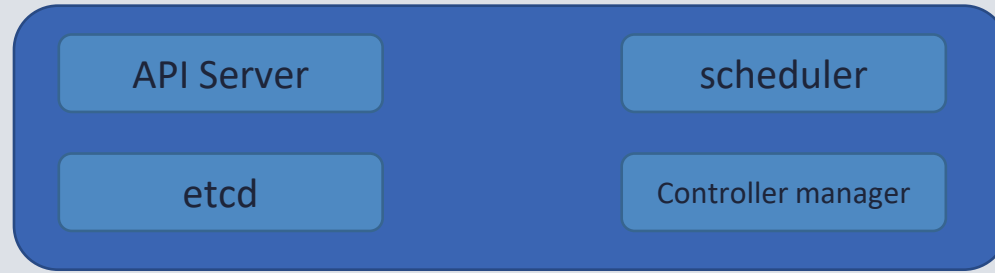


# Kubernetes AuthN & AuthZ

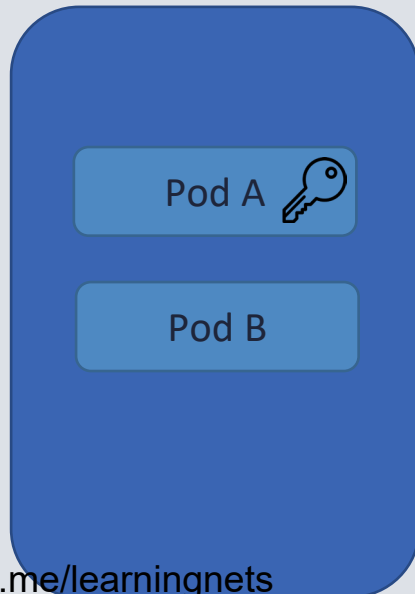
- Service accounts are mounted to pods, allowing them to authenticate with the API server.



## K8s control plane



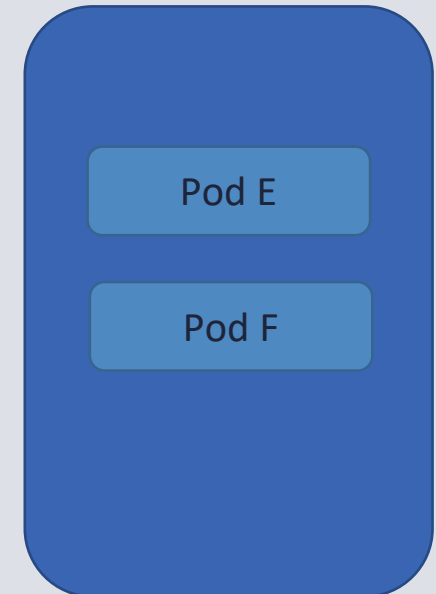
### Node 1



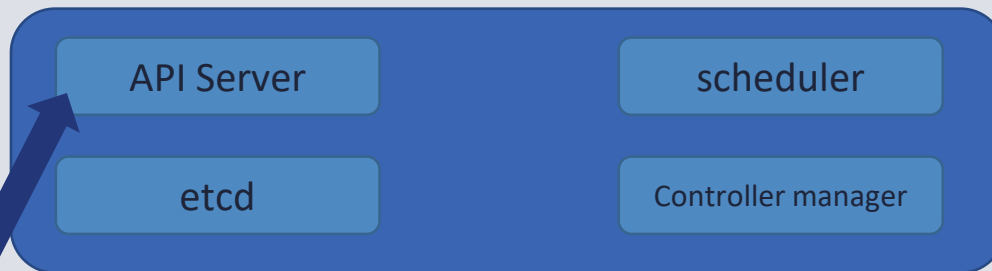
### Node 2



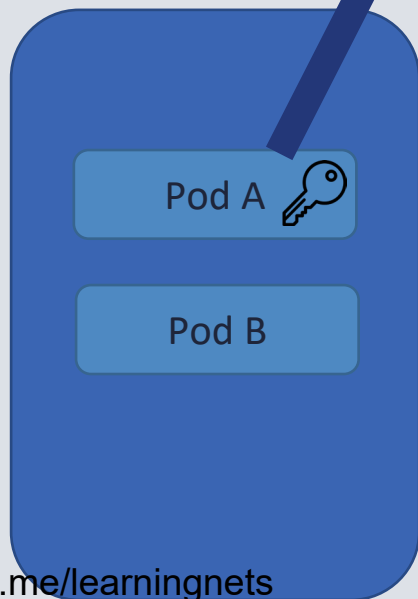
### Node 3



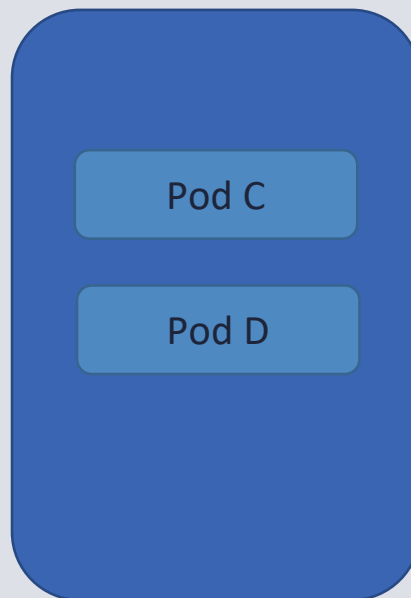
## K8s control plane



### Node 1



### Node 2



### Node 3



RSAConference™2023



# Inner-cluster lateral movement

# Inner-cluster lateral movement



Let's assume a pod is compromised

## K8s control plane



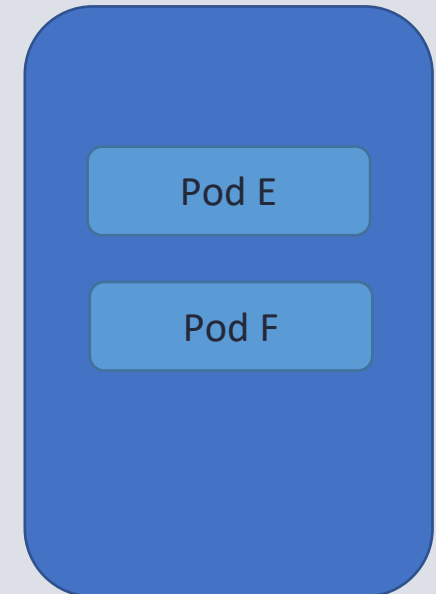
### Node 1



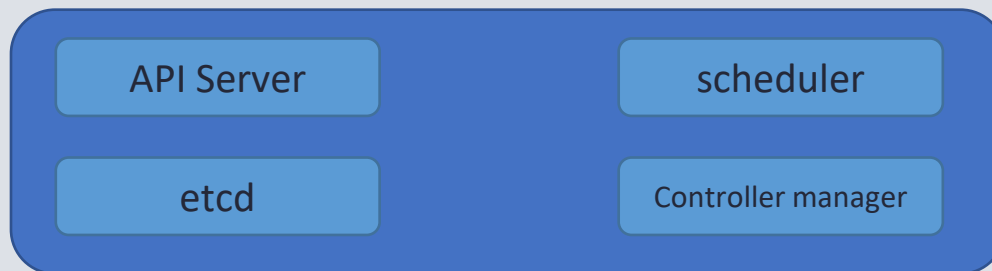
### Node 2



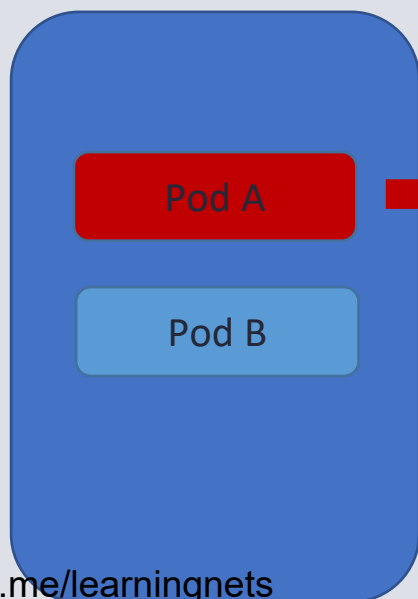
### Node 3



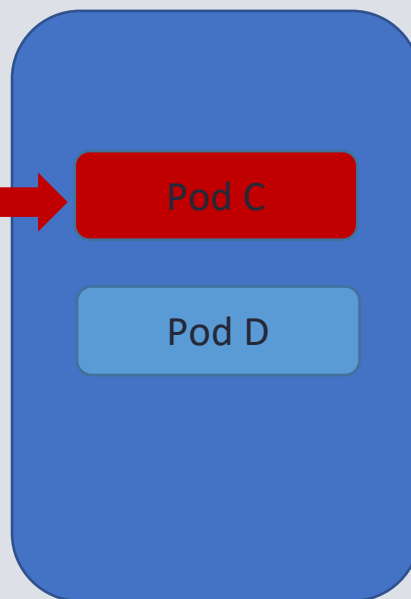
## K8s control plane



### Node 1



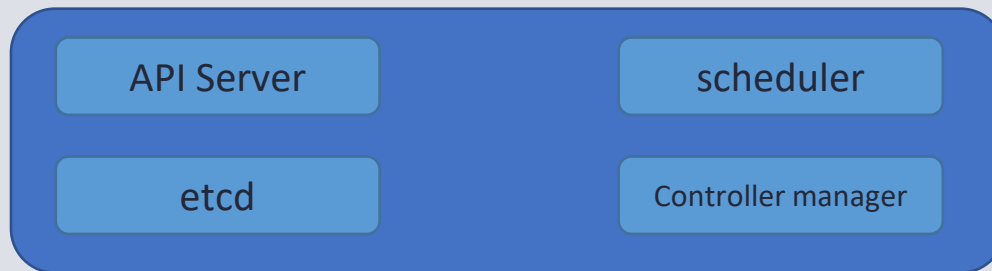
### Node 2



### Node 3



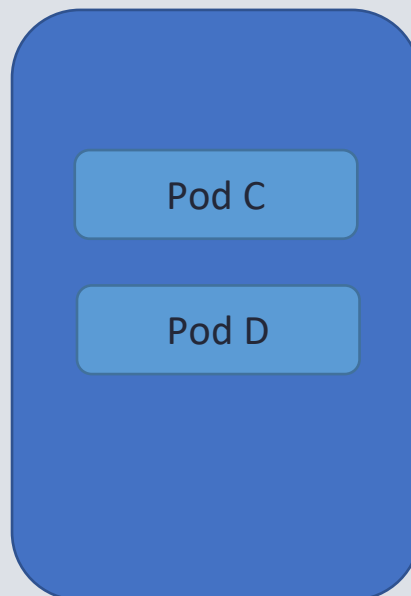
## K8s control plane



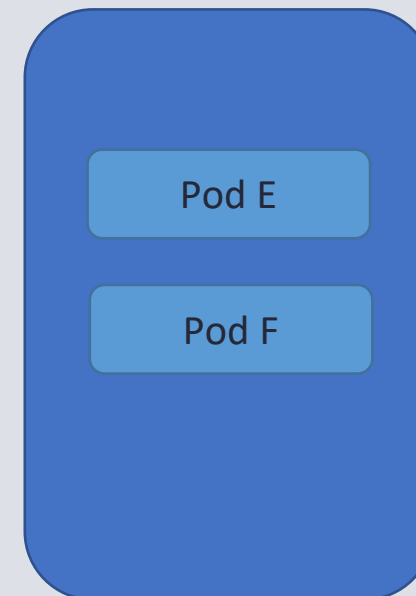
### Node 1



### Node 2



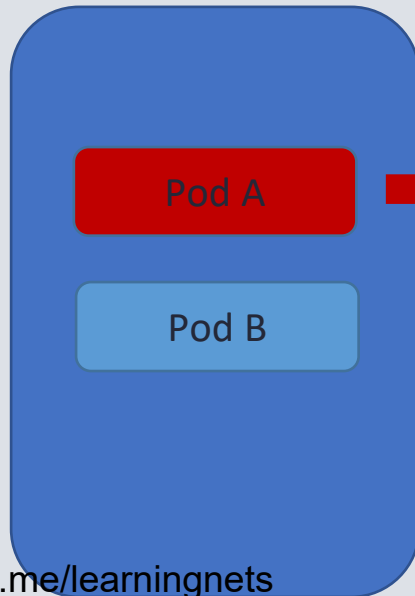
### Node 3



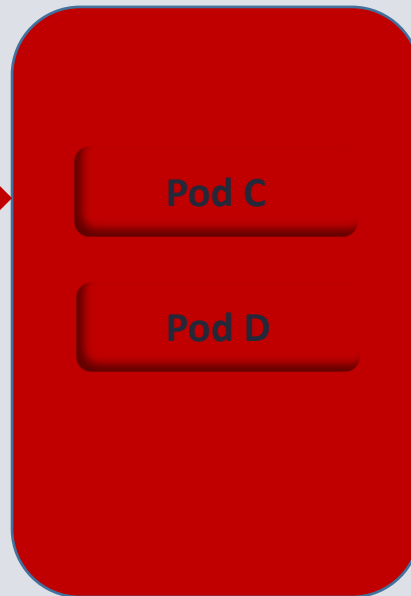
## K8s control plane



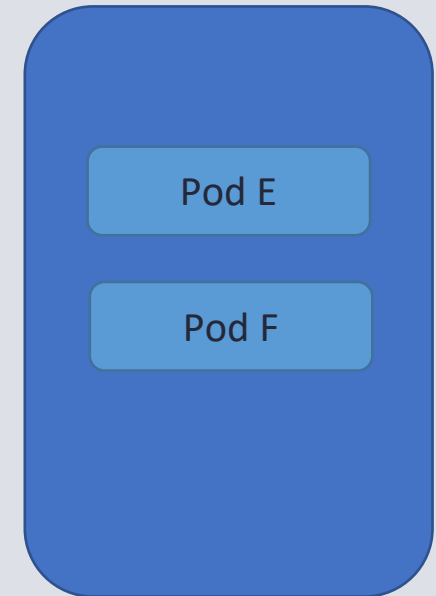
### Node 1



### Node 2



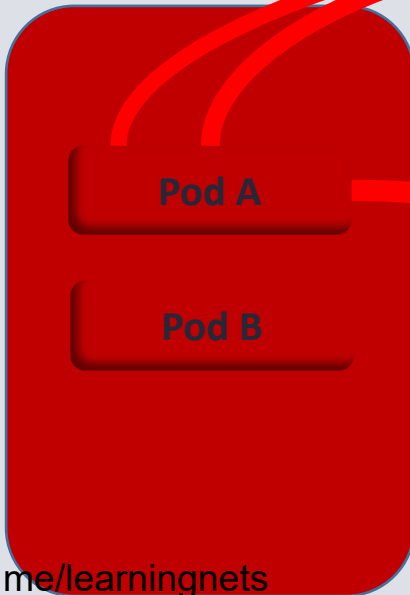
### Node 3



## K8s control plane



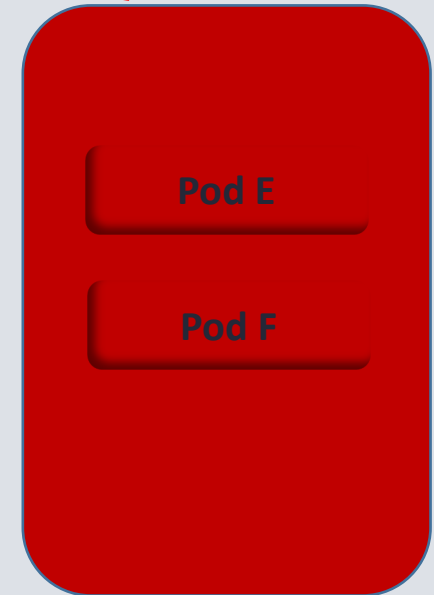
Node 1



Node 2



Node 3



# Inner-cluster lateral movement

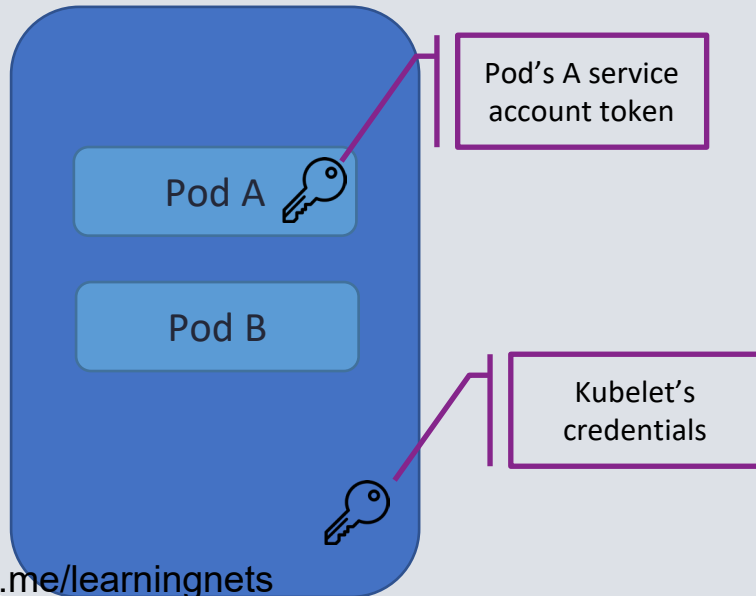


How can attackers leverage a compromised pod for cluster takeover?

## K8s control plane



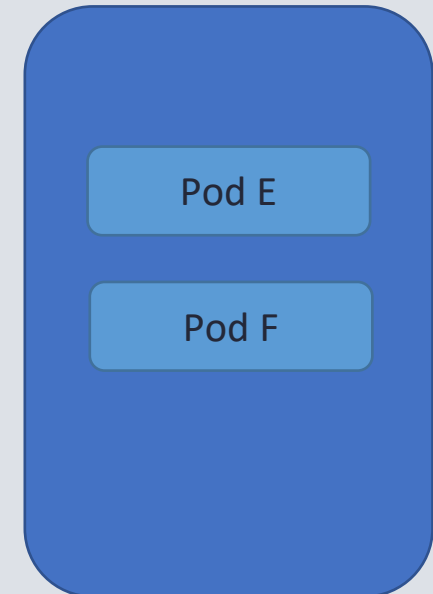
### Node 1



### Node 2



### Node 3



# Inner-cluster lateral movement

#RSAC

Stronger  
Together

How can attackers leverage a compromised pod for cluster takeover?

**Good news** – it becomes more difficult:

- **Read secrets permission isn't enough**
  - Newer versions of K8s don't create long-lived SA tokens as secret objects
- **Node takeover  $\neq$  cluster takeover**
  - Node authorizer + NodeRestriction admission controller limit the permissions of the nodes' identities

# Inner-cluster lateral movement

How can attackers leverage a compromised pod for cluster takeover?

**Good news** – it becomes more difficult:

- **Read secrets permission isn't enough**
  - Newer versions of K8s don't create long-lived SA tokens as secret objects
- **Node takeover  $\neq$  cluster takeover**
  - Node authorizer + NodeRestriction admission controller limit the permissions of the nodes' identities

**However**, common misconfigurations still allow it.

Let's see an example

# Inner-cluster lateral movement

## Example: Self-update permissions

#RSAC

Stronger  
Together

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-update
rules:
- apiGroups:
  - 'apps'
  resources:
  - 'deployments'
  verbs:
  - 'update'
  resourceNames:
  - 'my-app'
```

- We've observed cases in which applications had permissions to update themselves.
- This allows the applications to change their own configuration. For example: update their configuration to be privileged, change their service account, schedule to specific node(s).
- Effectively, this can lead to cluster takeover

## K8s control plane



Change the  
application  
configuration

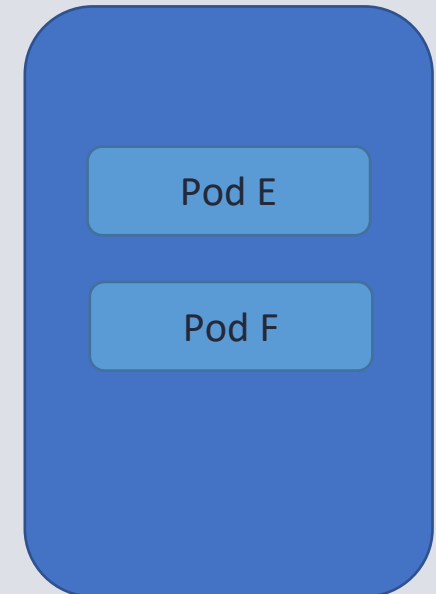
### Node 1



### Node 2



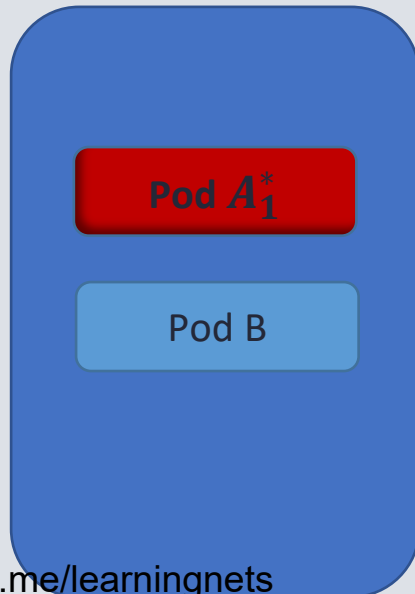
### Node 3



## K8s control plane



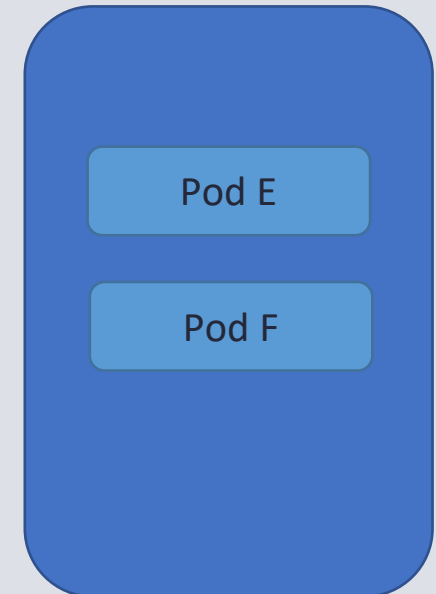
### Node 1



### Node 2



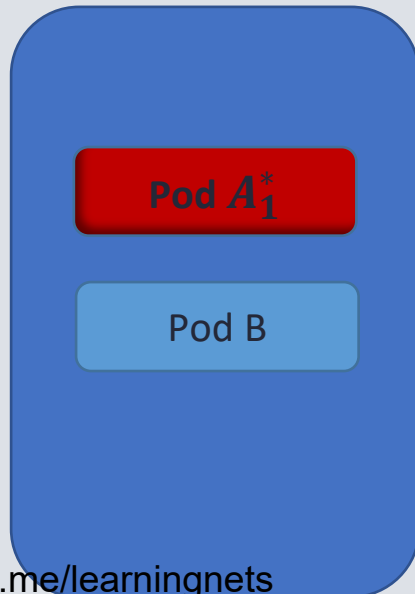
### Node 3



## K8s control plane



### Node 1



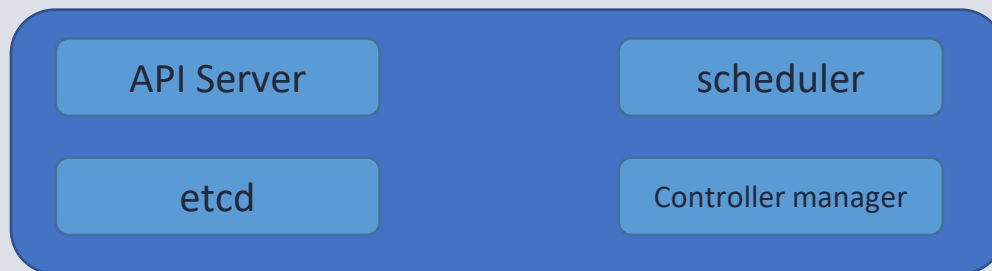
### Node 2



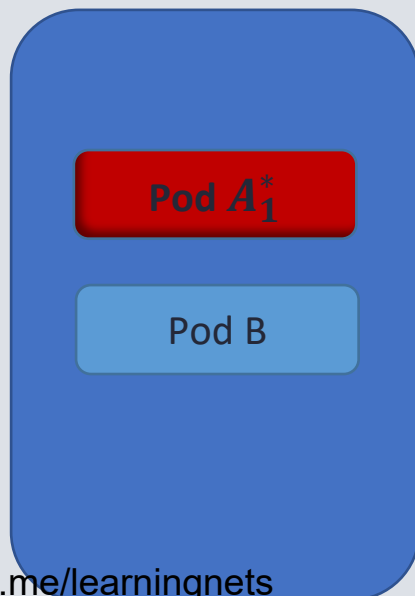
### Node 3



## K8s control plane



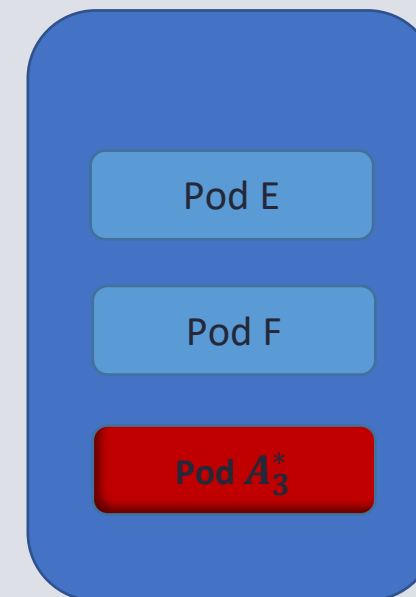
### Node 1



### Node 2



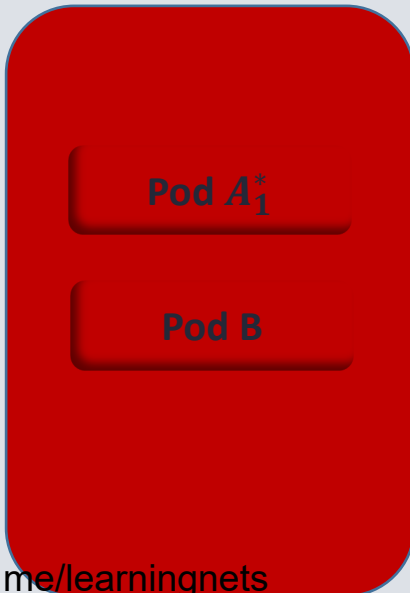
### Node 3



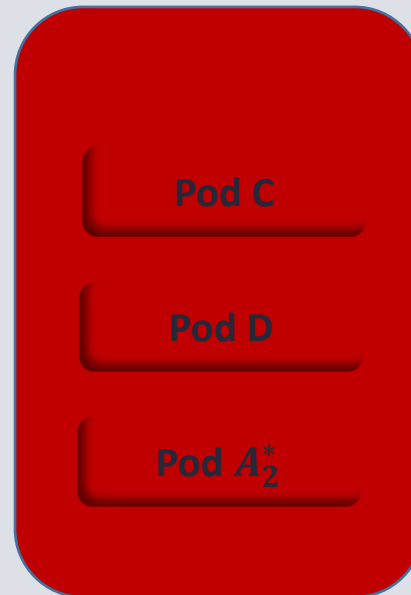
## K8s control plane



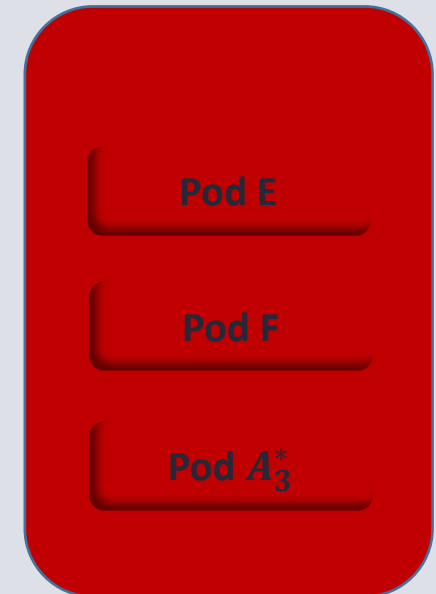
### Node 1



### Node 2



### Node 3



# Inner-cluster lateral movement

## Other sensitive permissions (partial list):

Permission	Allows to
Create pod\controllers	Use a privileged service account in a new pod
Update controller	Change the configuration to use a privileged service account
Create secrets & get\list secret	Create a new long-lived token for SA and read its value
Create serviceaccounts/token	Create a short-lived token for SA

**RSA**Conference™2023



**Stronger  
Together**

# Cluster-to-cloud lateral movement

# Cluster-to-cloud lateral movement

#RSAC

Stronger  
Together

- Workloads in Kubernetes may need access to cloud resources (for example: cloud storage \ cloud secret store etc.)
- Managed clusters use cloud services for the cluster operation.
- How do workloads in Kubernetes authenticate with the cloud?

# Cluster-to-cloud lateral movement

#RSAC

Stronger  
Together

We'll go over the following methods:

1. Storing cloud identity credential on the node
2. Direct access to IMDS
3. Indirect access to IMDS
4. Using OIDC (identity federation)

# Cluster-to-cloud lateral movement

## Storing cloud identity credential on the node



- Used to be the default authentication method in AKS in the past.
- In this method, each Kubernetes node stores a file with service principal (SPN) credentials. SPNs are Azure application identities.
- By default, this SPN has Contributor role for the node resource group.
- Users can bring their own SPN or grant more permissions to the SPN if their applications need access to more cloud resources. For example: add permissions to a cloud storage.


# Cluster-to-cloud lateral movement


## Storing cloud identity credential on the node



- Used to be the default authentication method in AKS in the past.
- In this method, each Kubernetes node stores a file with service principal (SPN) credentials. SPNs are Azure application identities.
- By default, this SPN has Contributor role for the node resource group.
- Users can bring their own SPN or grant more permissions to the SPN if their applications need access to more cloud resources. For example: add permissions to a cloud storage.

**→ Access to the node's FS = Contributor permission on the resource group**

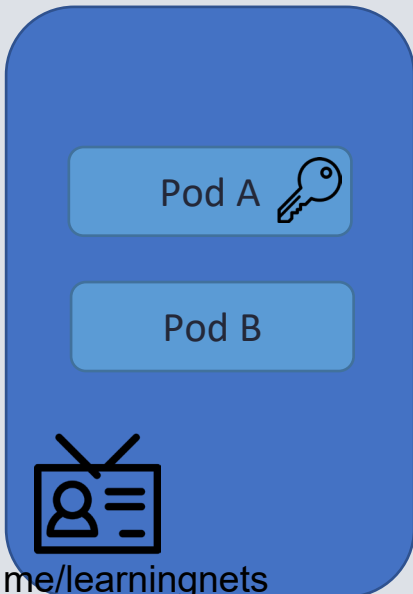
 Kubernetes Service Account

 Azure Service principal

## K8s control plane



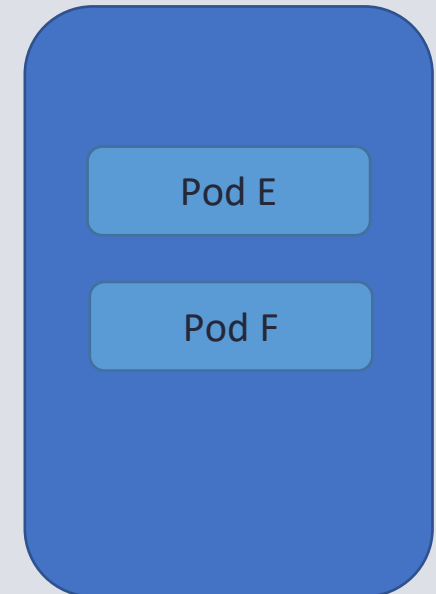
### Node 1





### Node 2



### Node 3



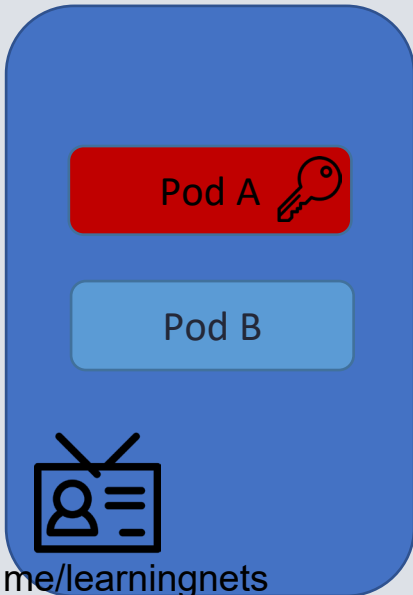
 Kubernetes Service Account

 Azure Service principal

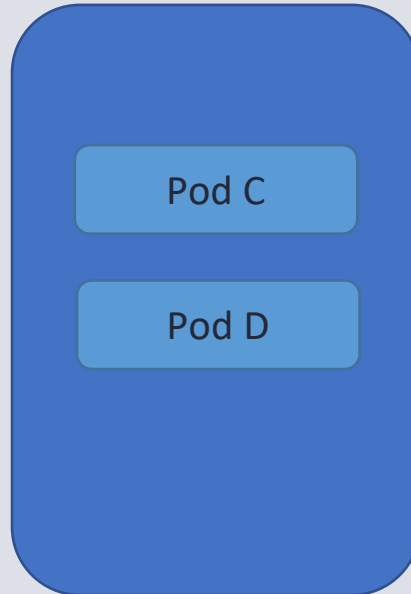
### K8s control plane



### Node 1




### Node 2



### Node 3



 Kubernetes Service Account

 Azure Service principal

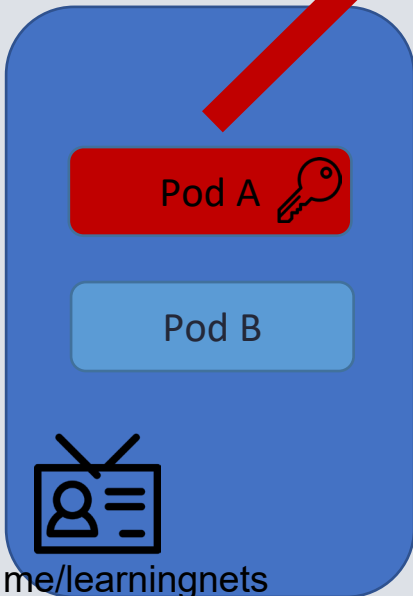
### K8s control plane



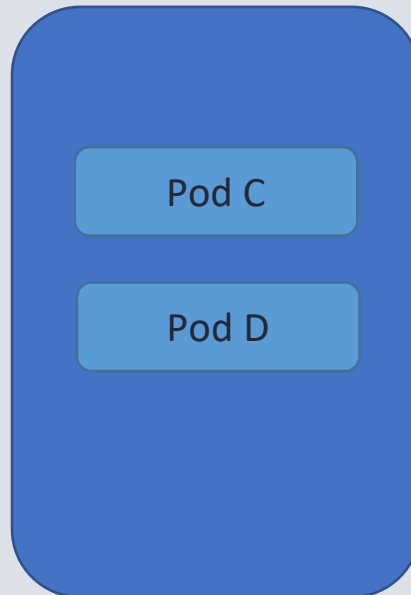
“Pod create” operation

In the pod config: mount the SPN into the container

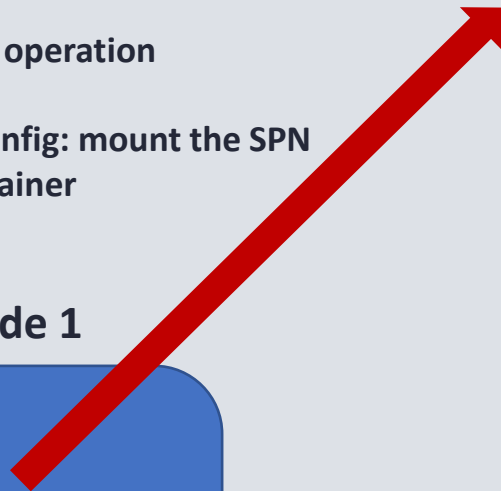
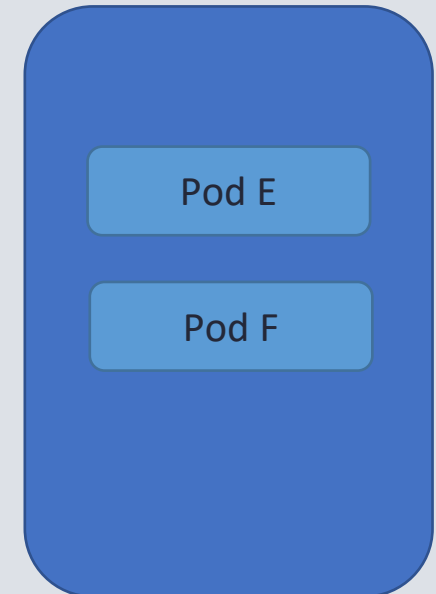
Node 1




Node 2



Node 3



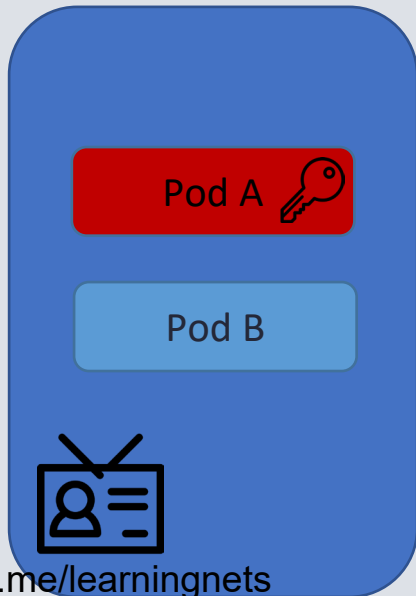
 Kubernetes Service Account

 Azure Service principal

### K8s control plane



### Node 1





### Node 2



### Node 3



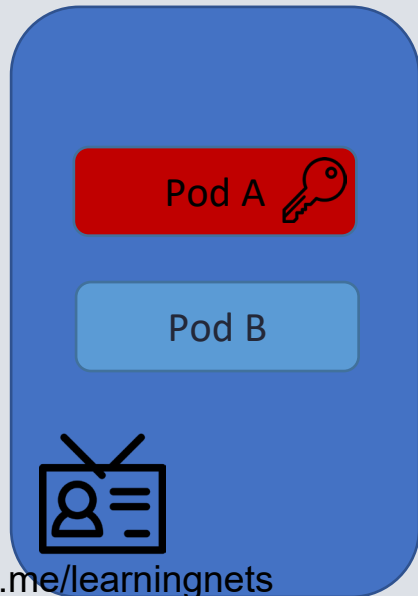
 Kubernetes Service Account

 Azure Service principal

### K8s control plane



### Node 1



### Node 2



### Node 3



# Cluster-to-cloud lateral movement

## Direct access to IMDS



- The metadata service is a special endpoint that is accessible to VMs, allowing them to retrieve information about the VM.
- Implemented by all major cloud providers.
- Metadata service allows retrieving tokens for the cloud identity that is attached to the VM:
  - Azure: `169.254.169.254/metadata/identity/oauth2`
  - AWS: `169.254.169.254/latest/meta-data/iam/security-credentials`
  - GCP: `metadata.google.internal/computeMetadata/v1/instance/service-accounts`
- Querying the metadata service doesn't require any authentication.

# Cluster-to-cloud lateral movement

## Direct access to IMDS



- In managed K8s clusters, the nodes are VM which can access to their metadata service.
- By default, pods can access to the metadata service of their nodes.
- Thus, pods can acquire tokens of cloud identities attached to the nodes.
- The permissions of the identities depend on the cloud provider and the specific environment.

# Cluster-to-cloud lateral movement

## Direct access to IMDS - AKS



- AKS uses several managed identities to operate the cluster.
- Users can change the default permissions of those identities, or alternatively attach additional managed identities to the nodes.

# Cluster-to-cloud lateral movement

## Direct access to IMDS - AKS

#RSAC

Stronger  
Together

Identity	Name	Use case	Default permissions	Bring your own identity
Control plane	AKS Cluster Name	Used by AKS control plane components to manage cluster resources including ingress load balancers and AKS managed public IPs, Cluster Autoscaler, Azure Disk & File CSI drivers	Contributor role for Node resource group	Supported
Kubelet	AKS Cluster Name-agentpool	Authentication with Azure Container Registry (ACR)	NA (for kubernetes v1.15+)	Supported
Add-on	AzureNPM	No identity required	NA	No
Add-on	AzureCNI network monitoring	No identity required	NA	No
Add-on	azure-policy (gatekeeper)	No identity required	NA	No
Add-on	azure-policy	No identity required	NA	No
Add-on	Calico	No identity required	NA	No
Add-on	Dashboard	No identity required	NA	No
Add-on	HTTPApplicationRouting	Manages required network resources	Reader role for node resource group, contributor role for DNS zone	No
Add-on	Ingress application gateway	Manages required network resources	Contributor role for node resource group	No
Add-on	omsagent	Used to send AKS metrics to Azure Monitor	Monitoring Metrics Publisher role	No
Add-on	Virtual-Node (ACIConnector)	Manages required network resources for Azure Container Instances (ACI)	Contributor role for node resource group	No
OSS project	aad-pod-identity	Enables applications to access cloud resources securely with Microsoft Azure Active Directory (Azure AD)	NA	Steps to grant permission at <a href="https://github.com/Azure/aad-pod-identity#role-assignment">https://github.com/Azure/aad-pod-identity#role-assignment</a> <sup>12</sup> .

# Cluster-to-cloud lateral movement

## Direct access to IMDS - EKS



- EKS uses EC2 Roles for the Kubernetes nodes.
- By default, the EC2 role has the policies:
  - AmazonEC2ContainerRegistryReadOnly - Pull permissions to the container registry.
  - AmazonEKSWorkerNodePolicy - Read permissions to the compute environment (EC2, VPC etc.)
  - AmazonEKS\_CNI\_Policy – Attach network interfaces and IPs to VMs
- Users can add more policies, if their containers require access to cloud resources.

# Cluster-to-cloud lateral movement

## Direct access to IMDS - GKE

#RSAC

Stronger  
Together

- GKE uses IAM service accounts to authenticate with the cloud.
- By default, all the VMs in a project, including the Kubernetes nodes, share a default SA.
- This SA has Editor role for the project.
- While the access scope limits the permissions, they are still powerful by default:

Security		
Sandbox with gVisor		Disabled
Service account ?		default
<b>Access scopes</b>		
	Service Control	Enabled
	Service Management	Read Only
	Stackdriver Logging API	Write Only
	Stackdriver Monitoring API	Full
	Stackdriver Trace	Write Only
	<b>Storage</b>	<b>Read Only</b>
GKE Metadata Server		Disabled
Integrity monitoring		Enabled
Secure boot		Disabled

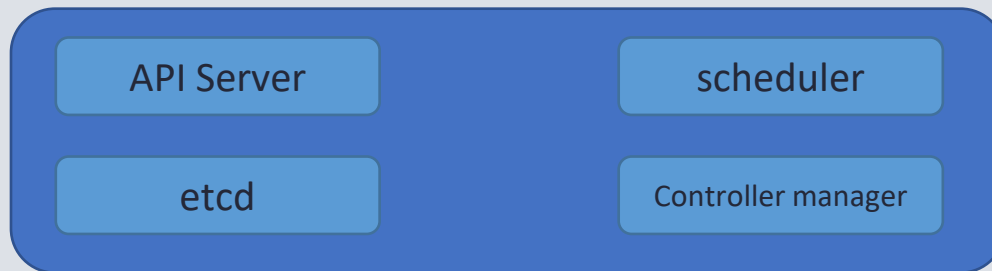
# Cluster-to-cloud lateral movement

Direct access to IMDS



How does lateral movement from the cluster to the cloud would look like?

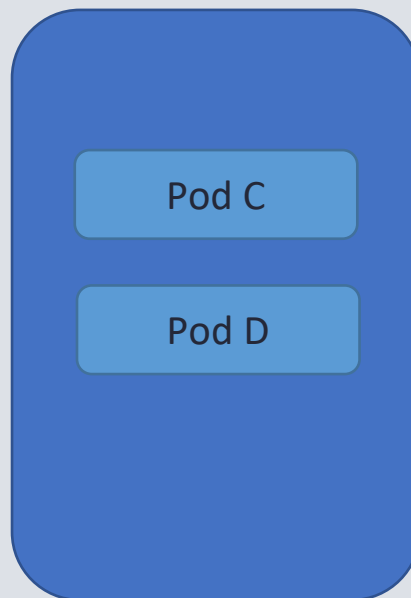
## K8s control plane



### Node 1



### Node 2



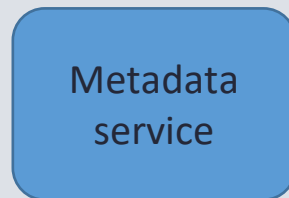
### Node 3



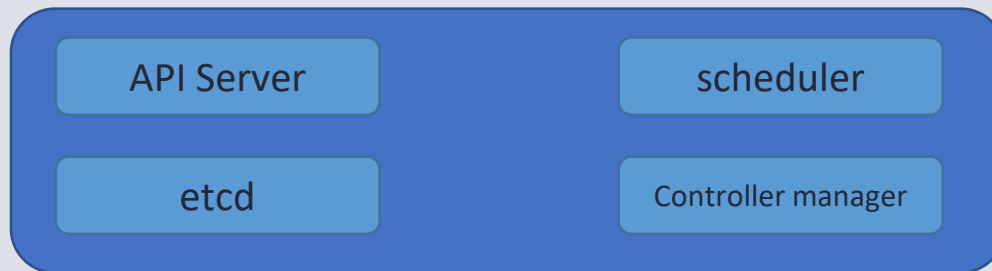
## K8s control plane



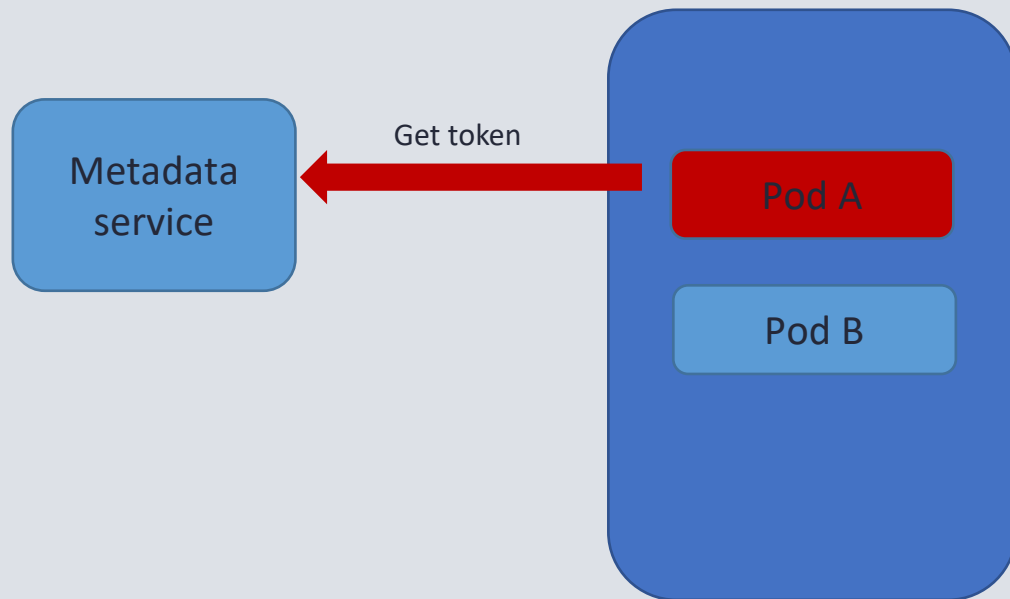
## Node 1



## K8s control plane



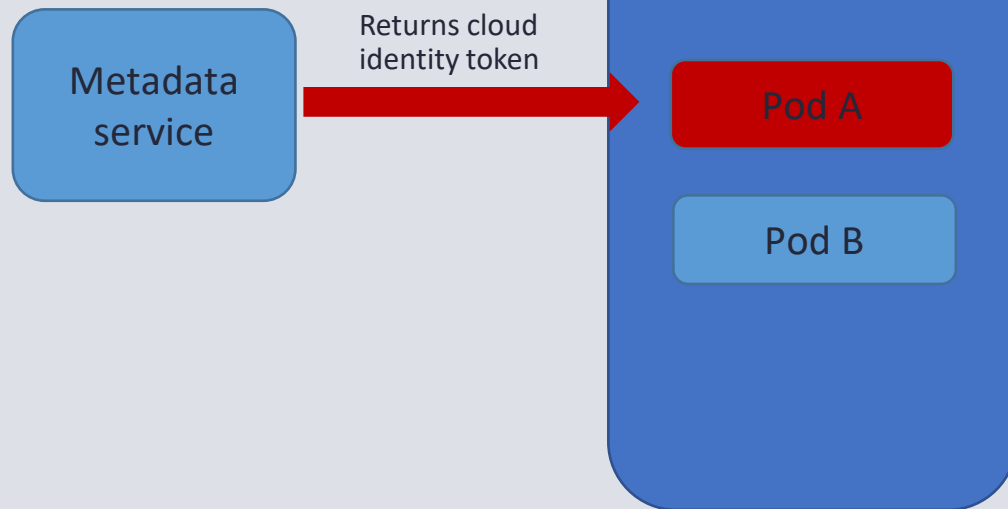
## Node 1

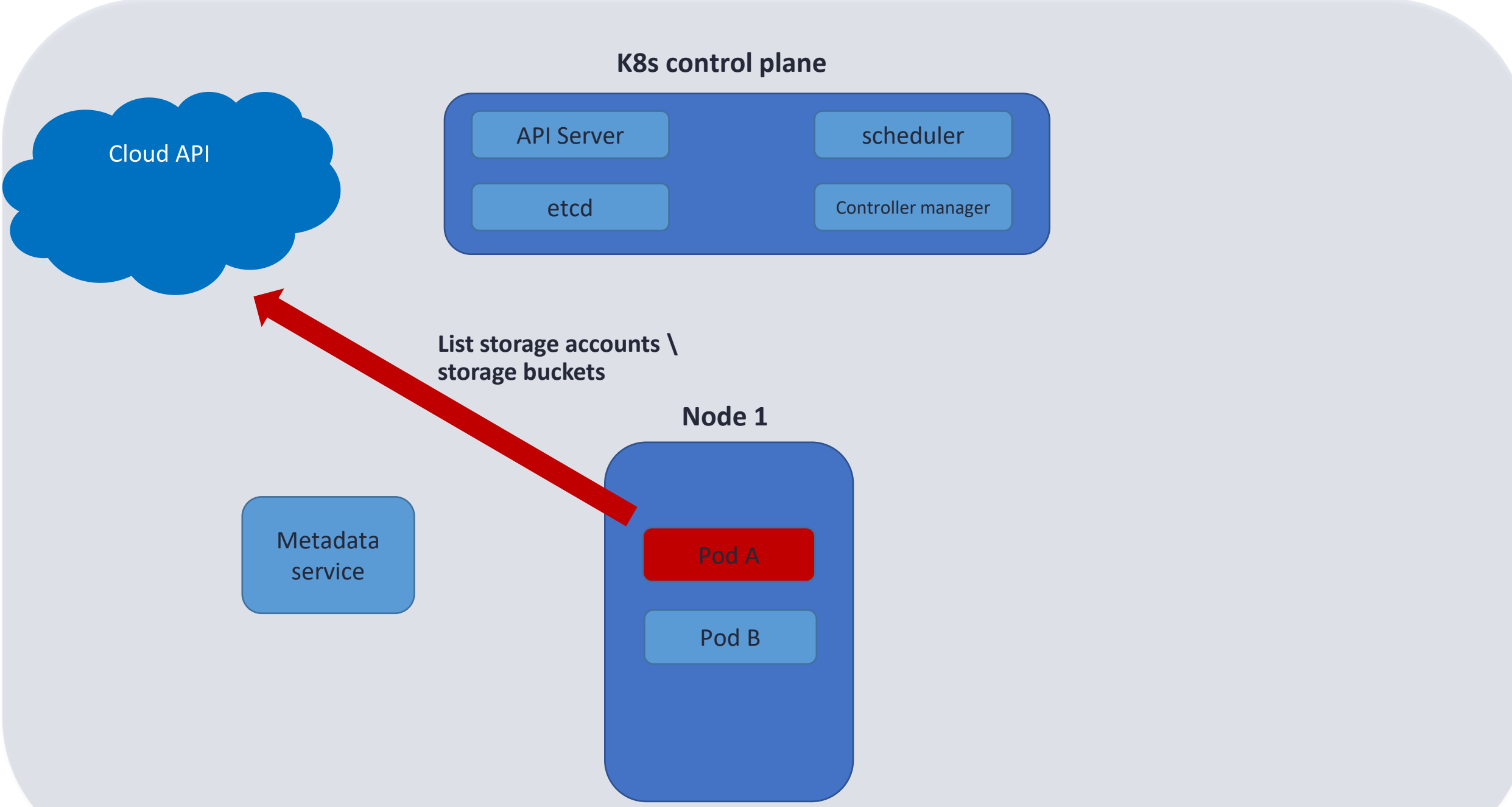


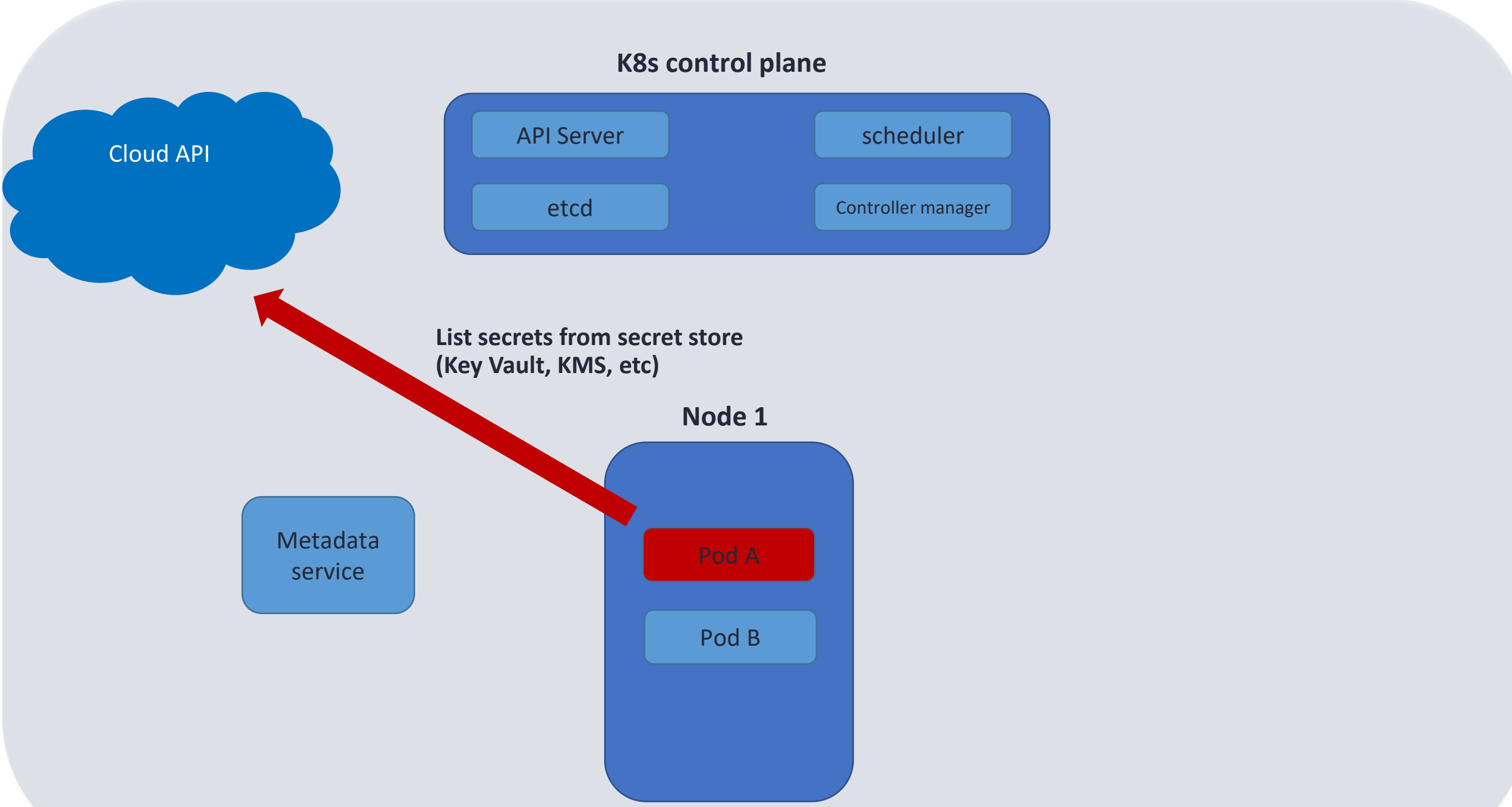
## K8s control plane

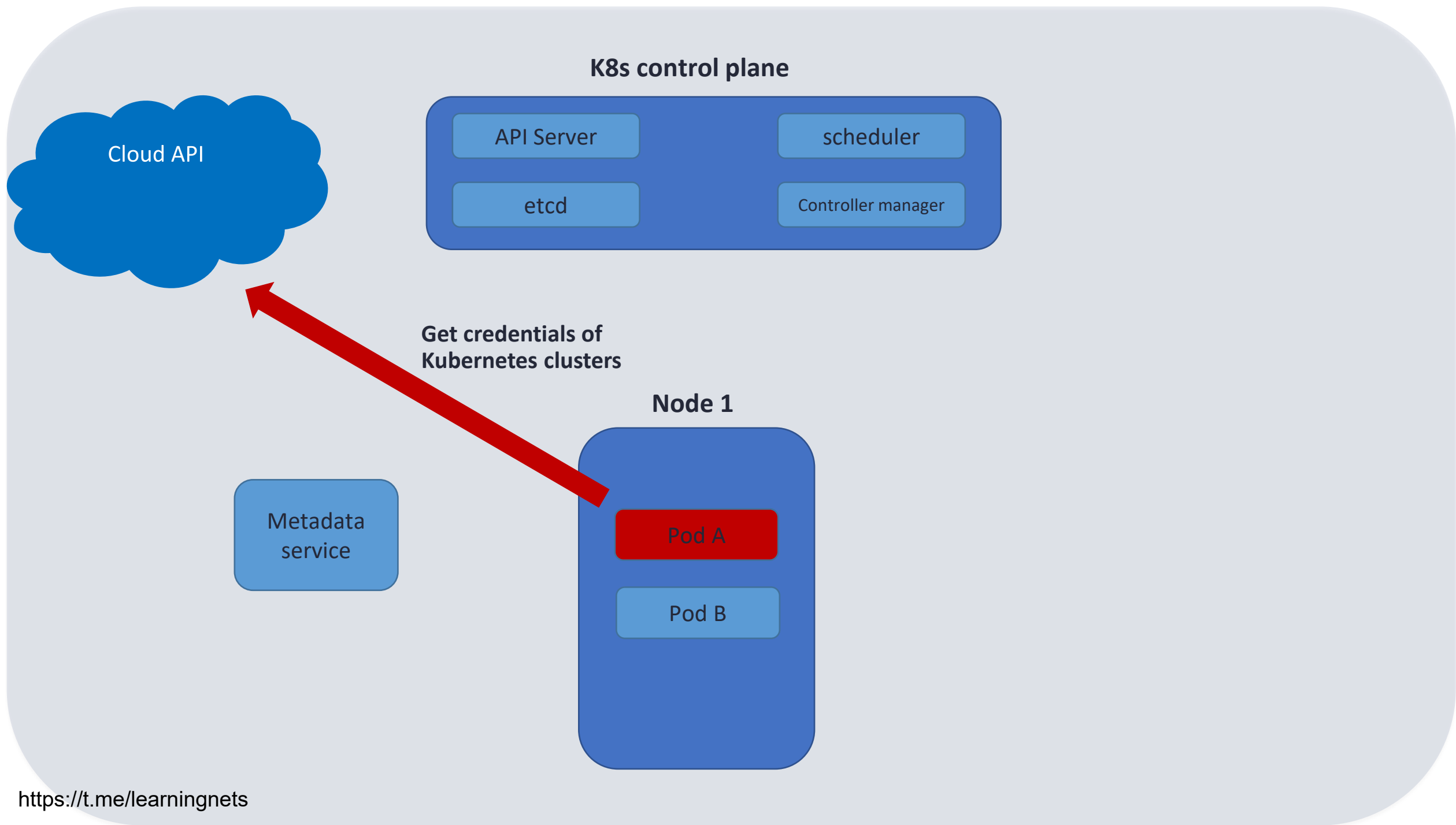


## Node 1









# Cluster-to-cloud lateral movement

Direct access to IMDS

#RSAC

Stronger  
Together

## The problem:

- Pods can freely access to their node's cloud identities.
- All pods share the same cloud identities (the node's identities).

## What we want:

- Allocate a specific identity to each pod (that needs access to cloud resources) with the minimal needed permissions.
- Make sure pods can only acquire tokens for their own identities.

# Cluster-to-cloud lateral movement

## Indirect access to IMDS

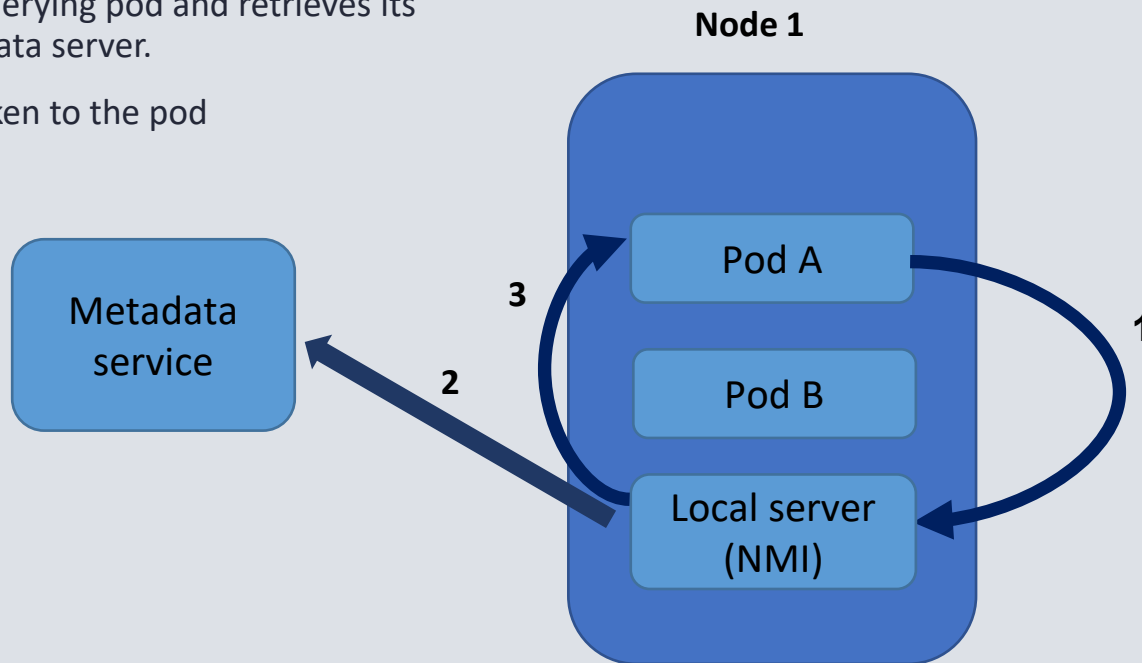


1. Users allocate different identities to the various applications in the cluster.
2. When pods query IMDS, the traffic is intercepted and redirected to a local server in the cluster.
3. The local server is K8s-aware, thus can identify the querying pod.
4. The local server queries IMDS on behalf of the pod and request the pod-specific identity.
5. This concept was implemented in Azure by AAD Pod Identity [recently deprecated].

## K8s control plane



1. Requests to the metadata server are intercepted and sent to the NMI pod (by modifying the IPTables of the node).
2. NMI recognizes the querying pod and retrieves its token from the metadata server.
3. NMI send back the token to the pod

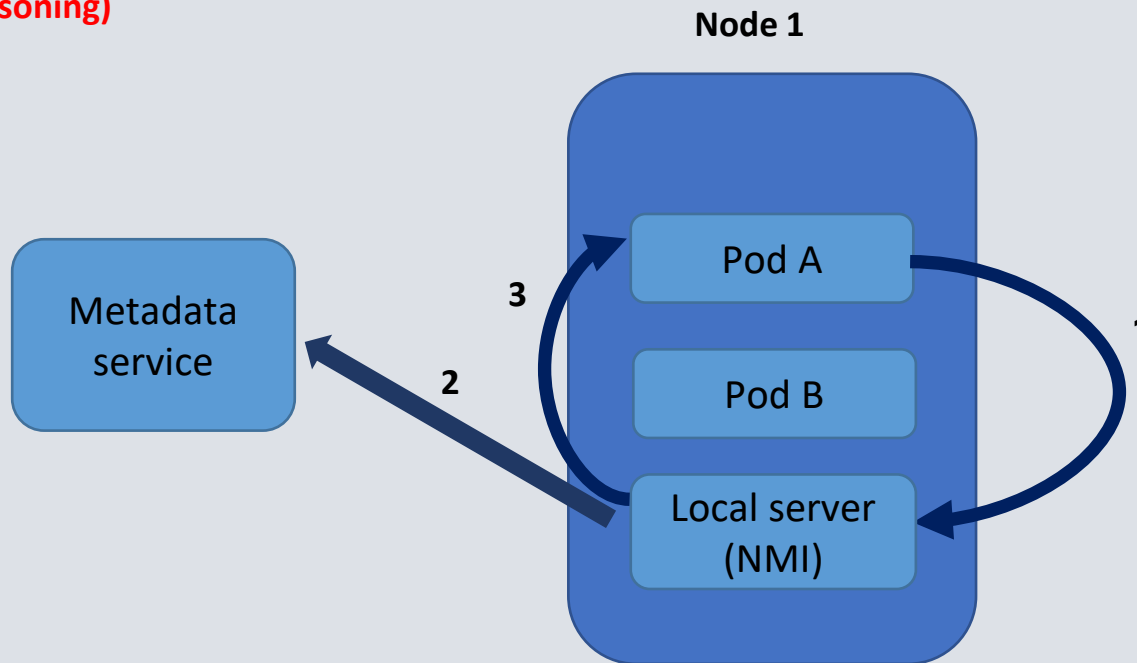


## K8s control plane



### Limitations

1. Works only for Linux containers (uses IPTables).
2. Not supported by all Kubernetes network configuration  
**(Prone to ARP poisoning)**



# Cluster-to-cloud lateral movement

## Using OIDC (identity federation)



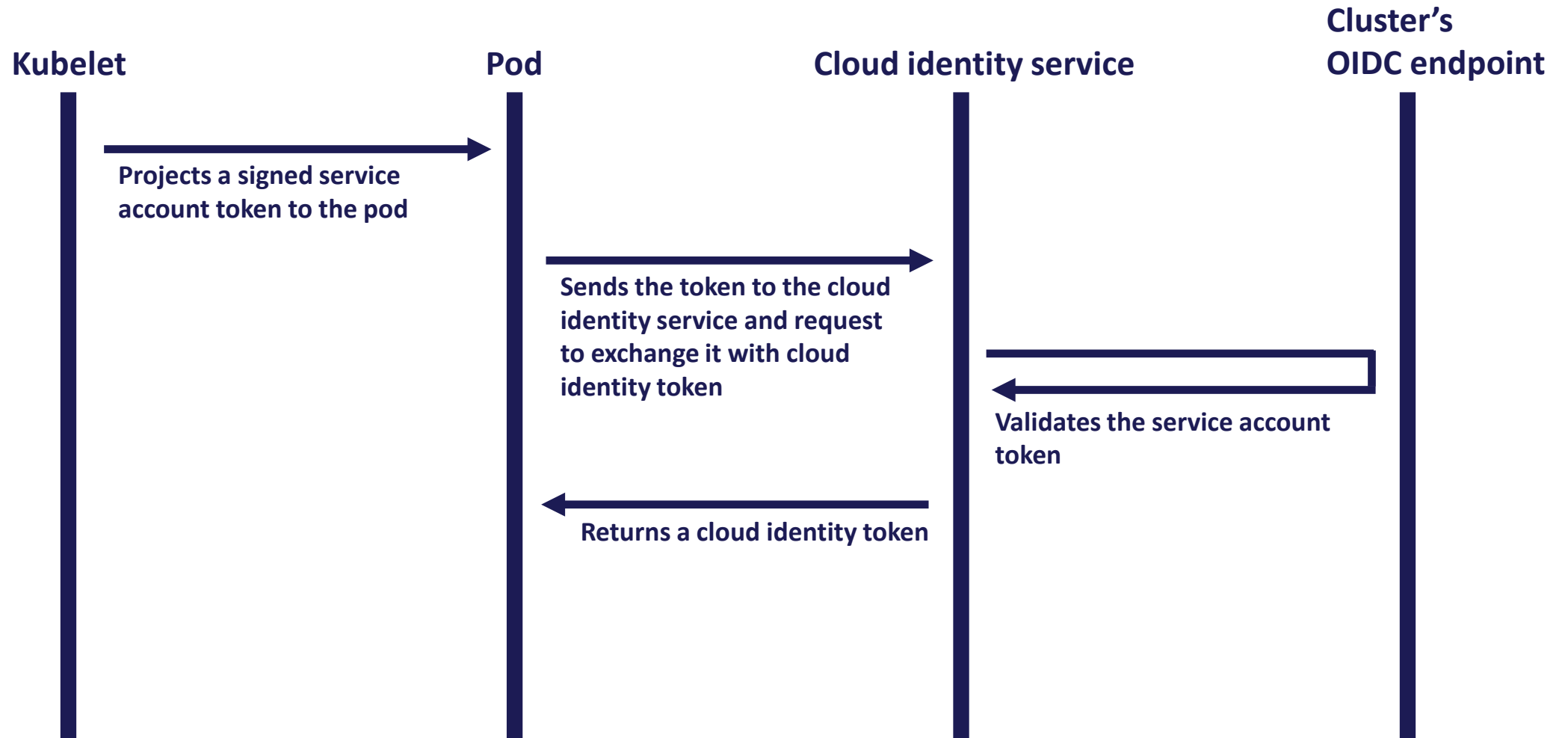
- Implemented by all major cloud providers:
  - Azure: AAD workload identity
  - AWS: IAM roles for service accounts (IRSA)
  - GCP: Workload identity
- The Kubernetes cluster is used as an OIDC identity provider (IdP).
- Trust relation is created such as the cloud identity service (e.g. AAD, AWS IAM, GCP IAM) trusts the service accounts issued by the K8s cluster.
- This trust relation allows applications in the cluster to exchange a K8s service account token with a cloud identity token.

# Cluster-to-cloud lateral movement

Using OIDC (identity federation)

#RSAC

Stronger Together

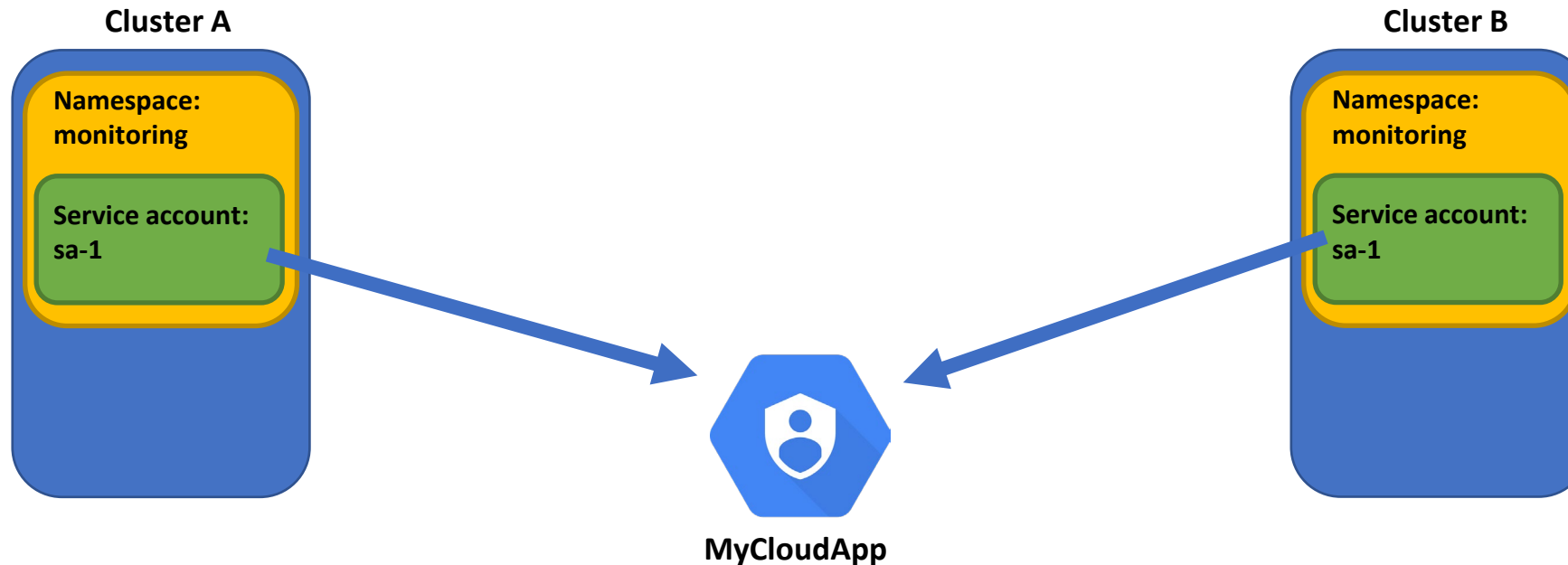


# Cluster-to-cloud lateral movement

## Using OIDC (identity federation) - GCP



- In GCP, there's a unified identity pool for the entire project.
- Meaning, there's a single binding of a K8s service account (namespace + SA name) to a cloud identity.



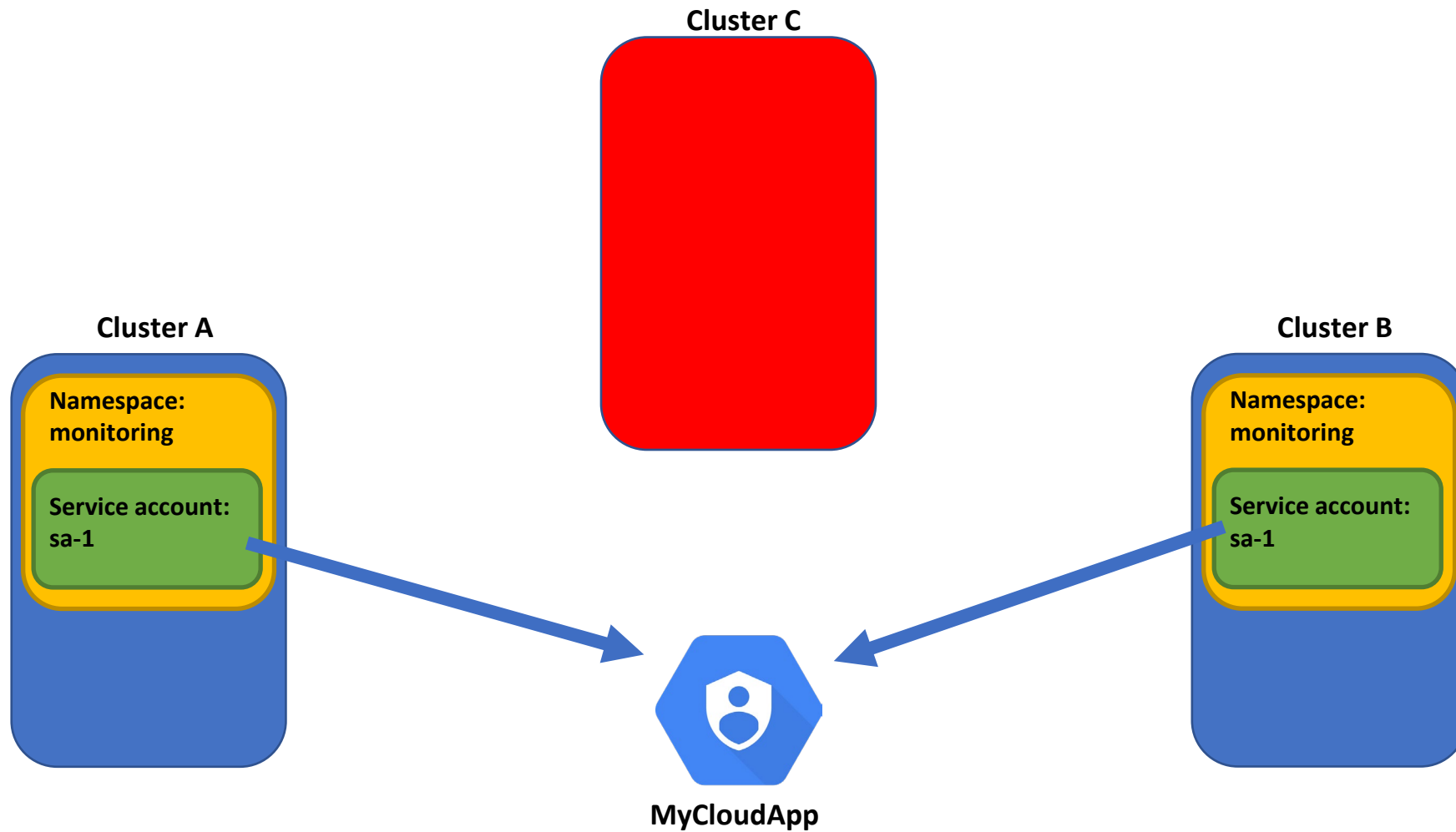
```
gcloud iam service-accounts add-iam-policy-binding MyCloudApp@My-GCP-Project.iam.gserviceaccount.com \
--role roles/iam.workloadIdentityUser \
--member "serviceAccount:My-GCP-Project.svc.id.goog[monitoring/sa-1]"
```

# Cluster-to-cloud lateral movement

Using OIDC (identity federation) - GCP

#RSAC

Stronger Together

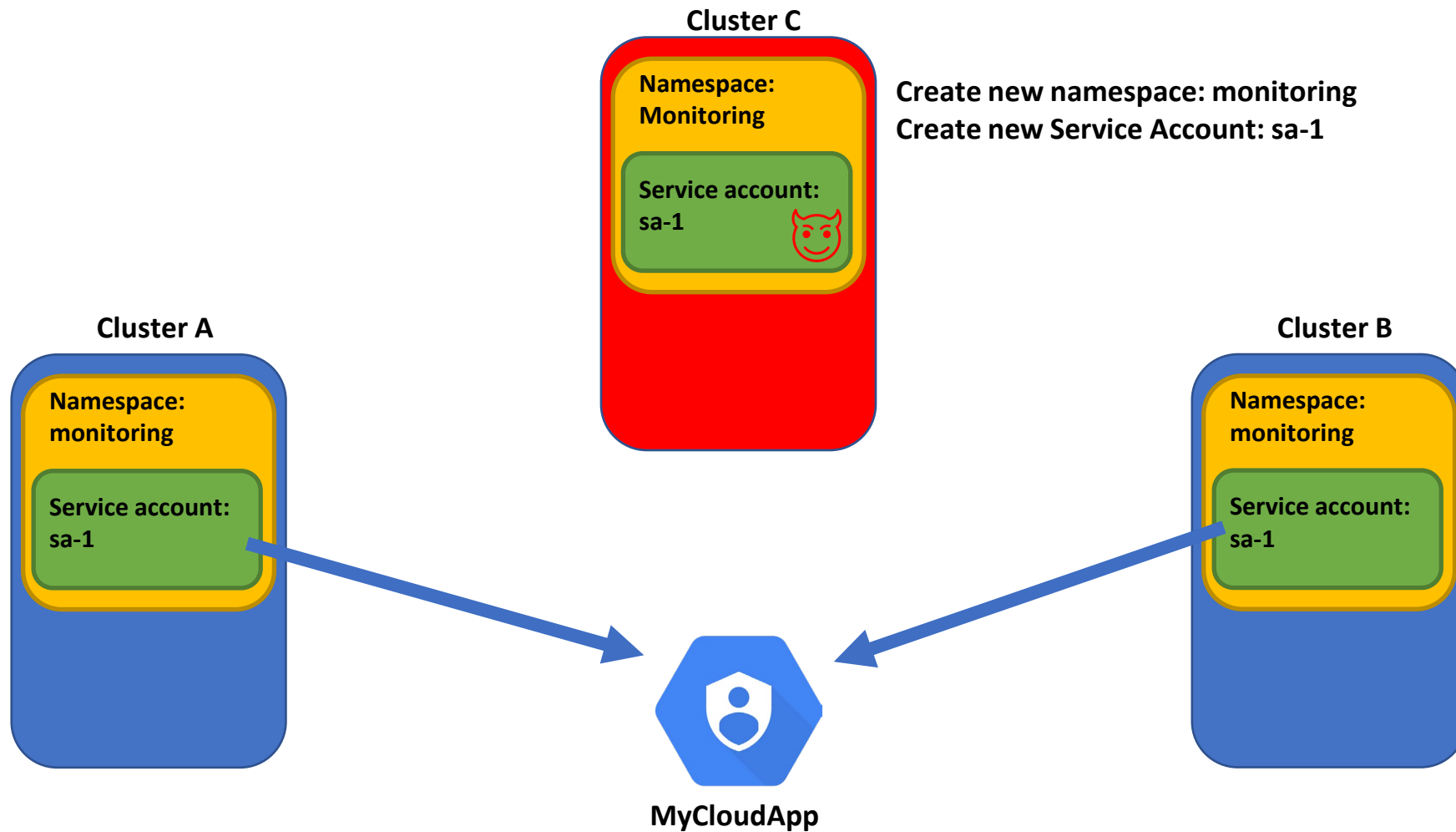


# Cluster-to-cloud lateral movement

Using OIDC (identity federation) - GCP

#RSAC

Stronger Together



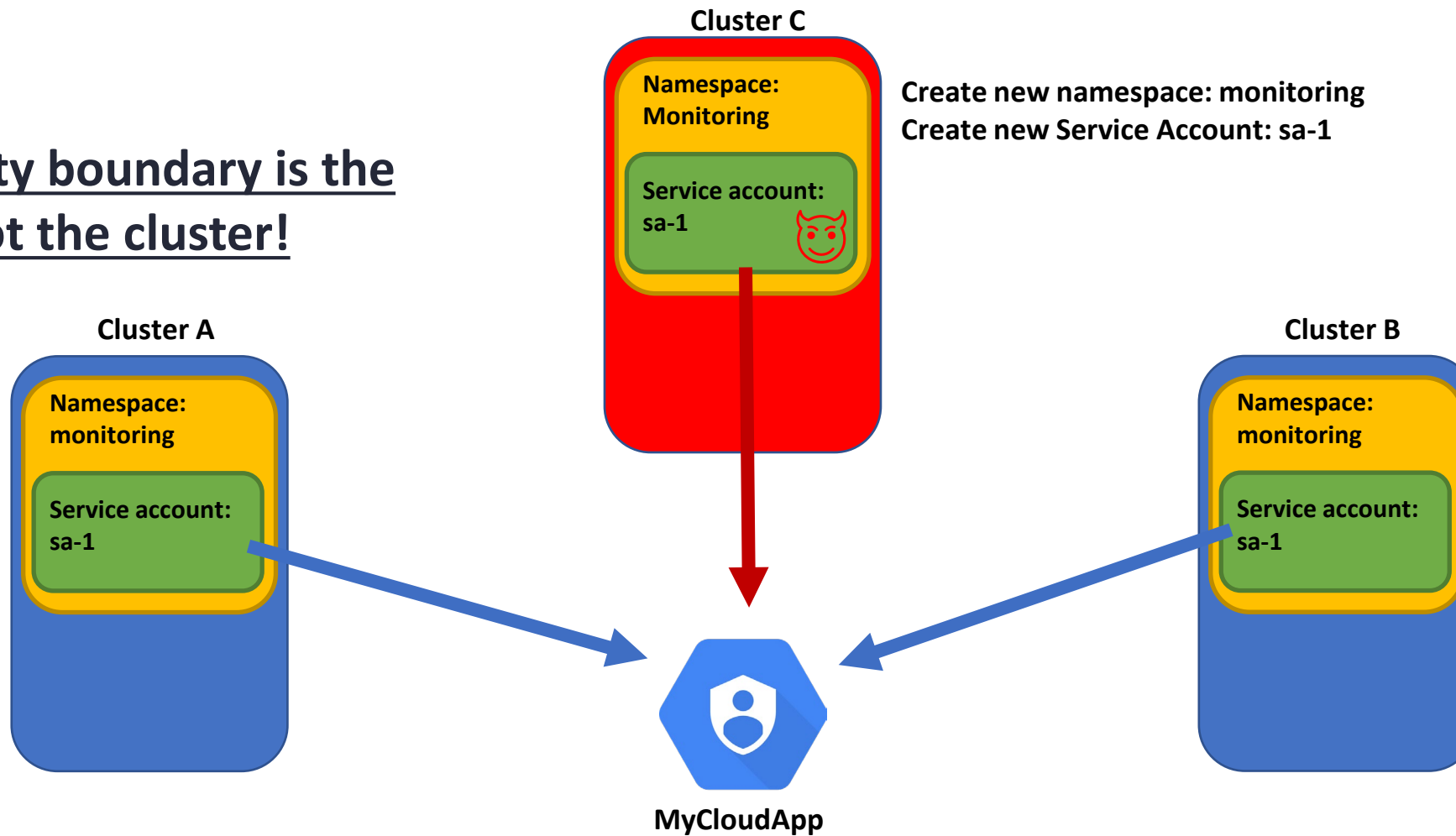
# Cluster-to-cloud lateral movement

Using OIDC (identity federation) - GCP

#RSAC

Stronger Together

The security boundary is the project, not the cluster!



RSAConference™2023



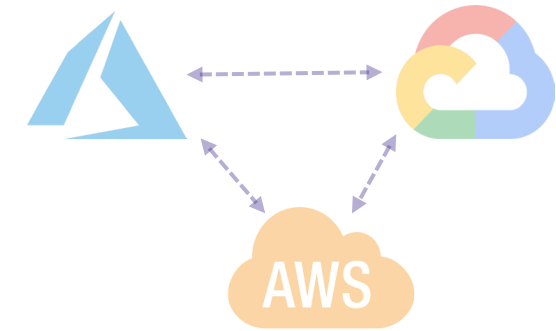
**Stronger  
Together**

# Cross-cloud lateral movement

# Cross-cloud lateral movement

## Possible cross-cloud lateral movement scenarios:

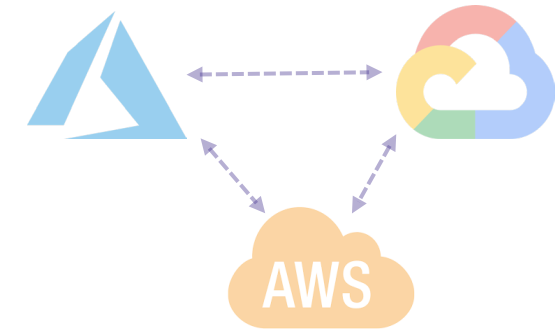
- Multi-cloud Kubernetes clusters
- Multi-cloud workload authentication
- Multi-cloud supply-chain attack



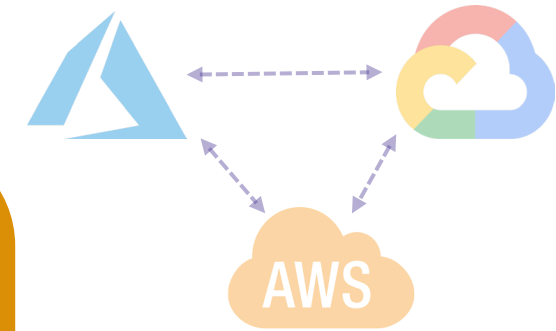
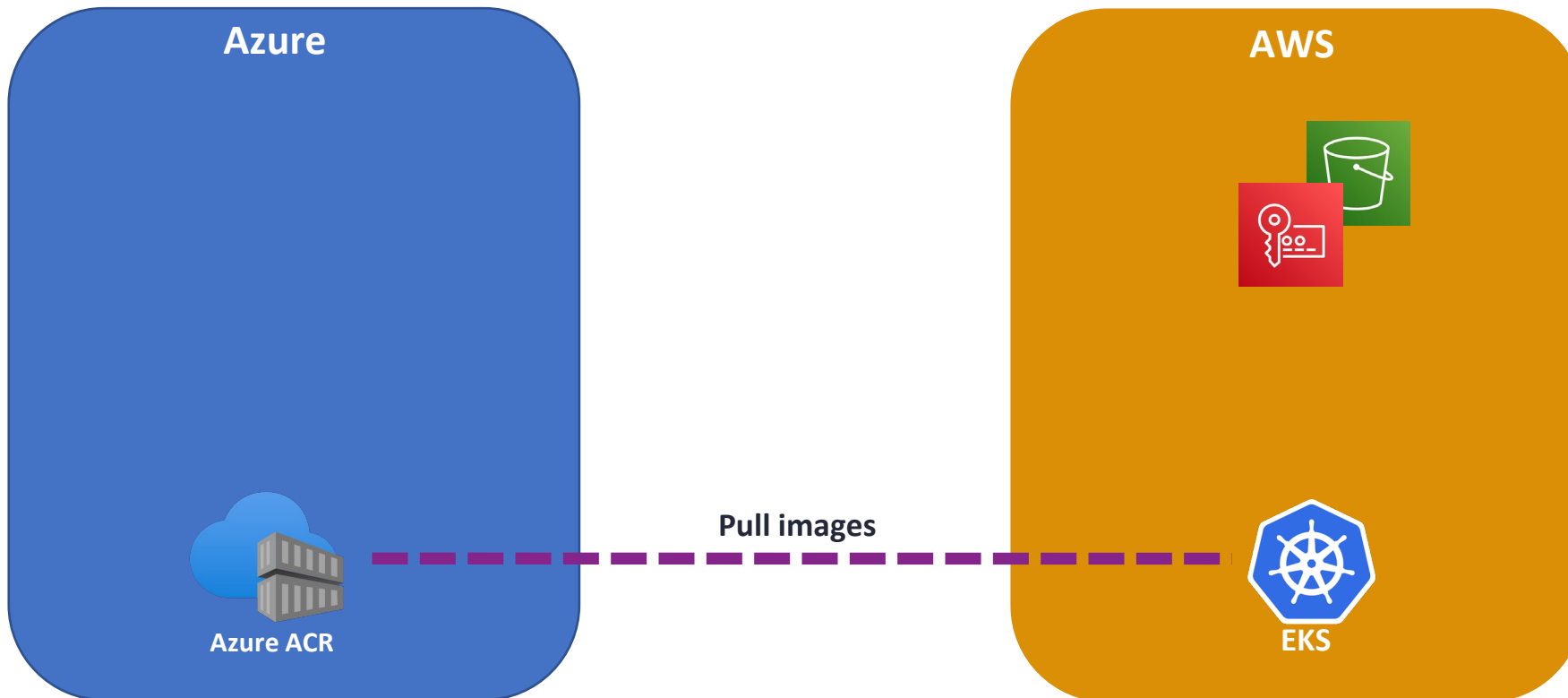
# Cross-cloud lateral movement

## Possible cross-cloud lateral movement scenarios:

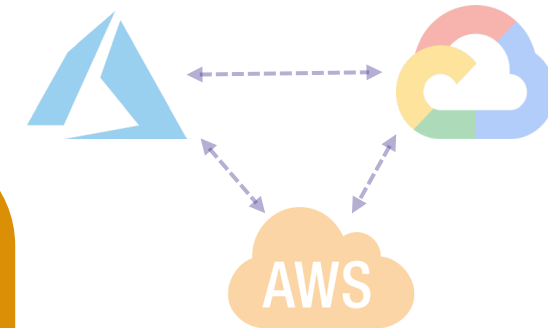
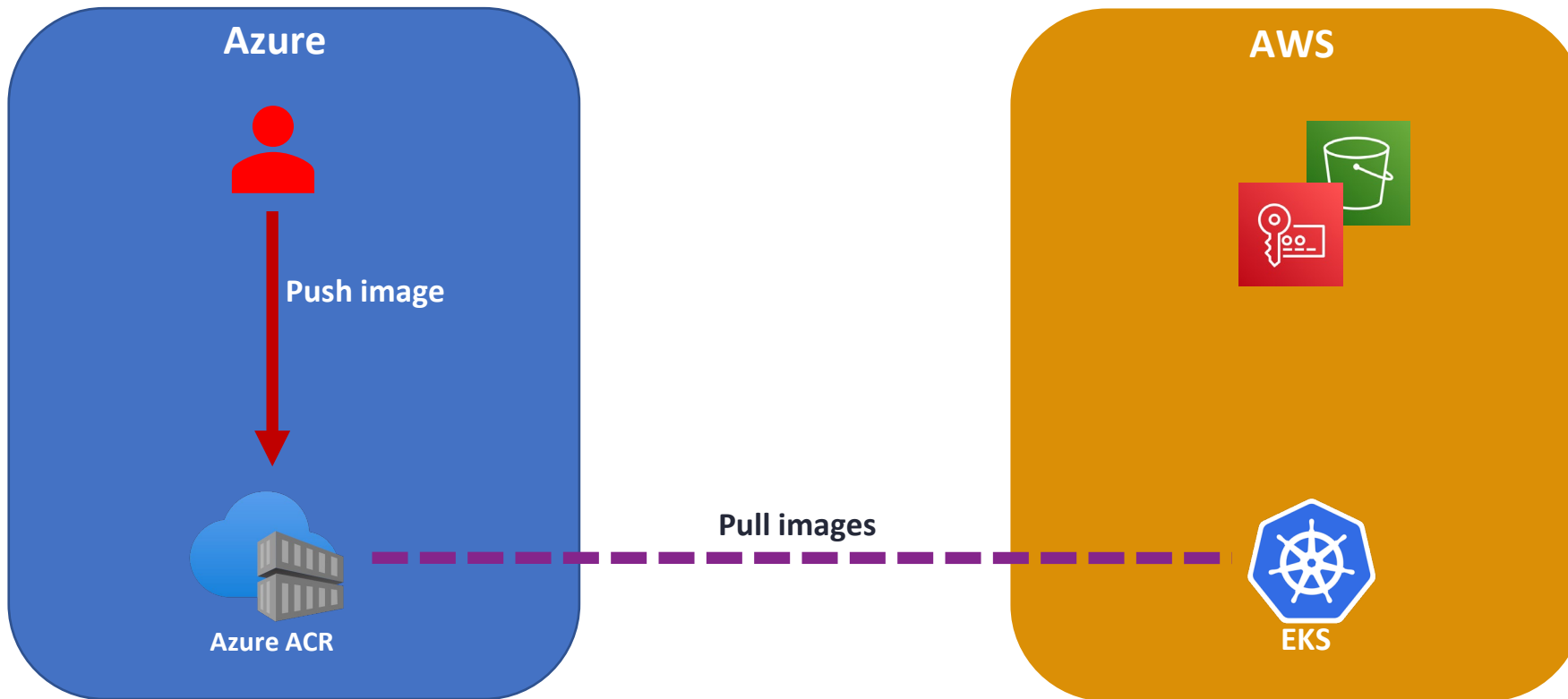
- Multi-cloud Kubernetes clusters
- Multi-cloud workload authentication
- **Multi-cloud supply-chain attack**



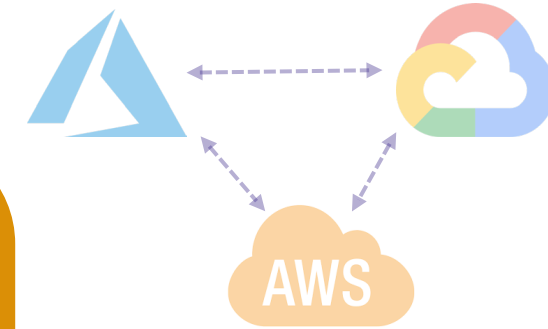
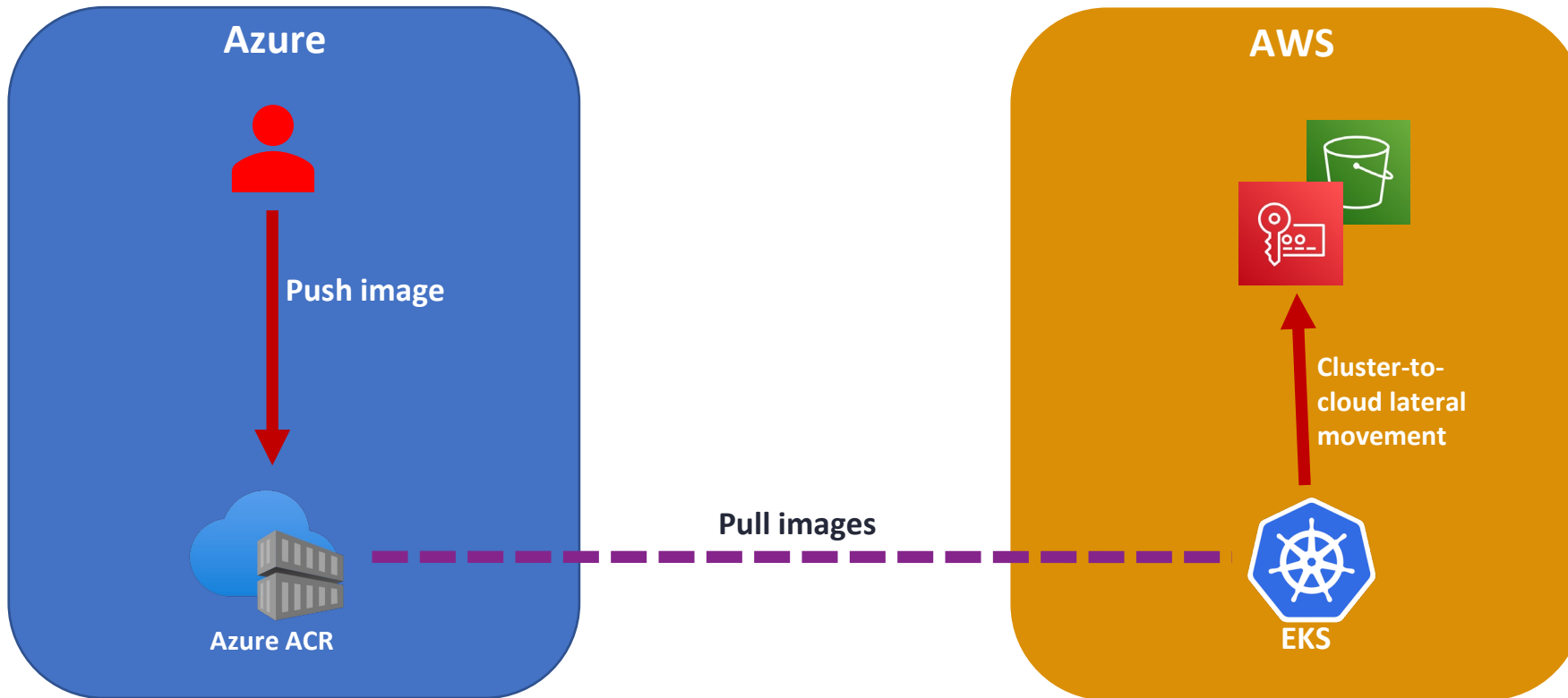
# Cross-cloud lateral movement



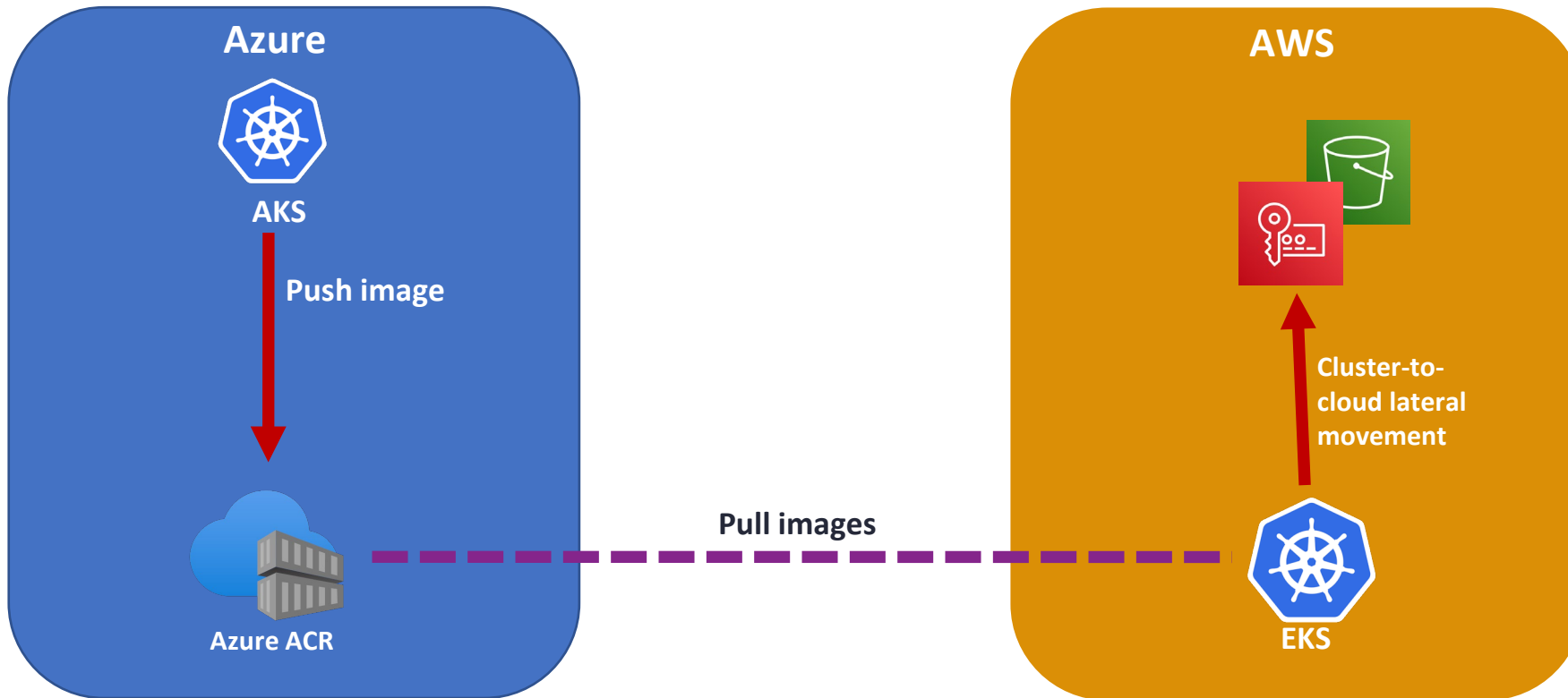
# Cross-cloud lateral movement



# Cross-cloud lateral movement



# Cross-cloud lateral movement



# What have we seen so far?



- Inner-cluster lateral movement:
  - Example: Self-update permissions that lead to cluster-takeover
  - Additional permissions can lead to cluster takeover (partial list)
- Cluster-to-cloud lateral movement:
  - Storing cloud identity credential on the node
  - Direct access to IMDS
  - Indirect access to IMDS
  - Using OIDC (identity federation)
- Cross-cloud lateral movement:
  - Multi-cloud supply-chain attack

**RSA**Conference™2023



**Stronger  
Together**

# Detections & Mitigations

# Detections



## Kubernetes control plane

**Monitor suspicious activity in the cluster using K8s Audit log (kube-audit). Examples:**

1. Deployment of abnormal images
2. Pods with suspicious configurations (sensitive volume mounts, privileged etc.)
3. Reconnaissance activity (for example: SelfSubjectRulesReview API call).
4. Sensitive API calls, such as “get secret”



## Cloud provider control plane

**Monitor suspicious activity of cloud identities used by K8s workloads\nodes. Examples:**

1. Abnormal behavior of cloud identities. Usually, the cloud identities used by the workloads have a consistent behavior.
2. Suspicious access to sensitive cloud services (e.g. storage, secret store etc.)

# Detections



## Kubernetes control plane

Monitor suspicious activity in the cluster using K8s Audit log (kube-audit). Examples:

1. Deployment of abnormal images
2. Pods with suspicious configurations (sensitive volume mounts, privileged etc.)
3. Reconnaissance activity (for example: SelfSubjectRulesReview API call).
4. Sensitive API calls, such as “get secret”

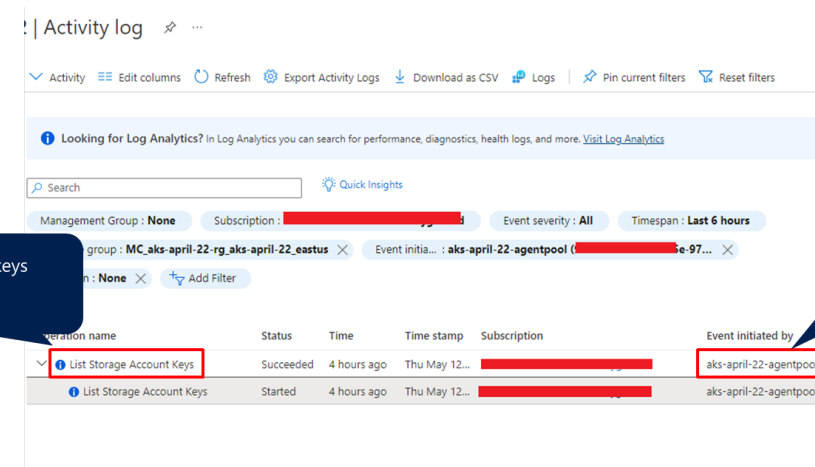


## Cloud provider control plane

Monitor suspicious activity of cloud identities used by K8s workloads\nodes. Examples:

1. Abnormal behavior of cloud identities. Usually, the cloud identities used by the workloads have a consistent behavior.
2. Suspicious access to sensitive cloud services (e.g. storage, secret store etc.)

Azure: Activity Log  
AWS: CloudTrail  
GCP: Cloud Audit Logs



List storage keys

Managed identity used by Kubernetes node

# Mitigations



- In December, a new version of the Threat Matrix for Kubernetes was released (v3): an open-source knowledge base of attacking techniques of K8s.
- The new version includes now also mitigation techniques.
- <http://aka.ms/KubernetesThreatMatrix>

# Mitigations

#RSAC

Stronger Together

## Tactics

- Initial Access >
- Execution >
- Persistence >
- Privilege Escalation >
- Defense Evasion >
- Credential Access >
- Discovery >
- Lateral Movement >
- Collection >
- Impact >

## Tactics

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud resources	Images from a private registry	Data destruction
Compromised image In registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Collecting data from pod	Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Application credentials in configuration files	Exposed sensitive interfaces	Application credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container	Container service account			Access managed identity credentials	Instance Metadata API	Writable hostPath mount		
	Sidecar injection	Static pods			Malicious admission controller		CoreDNS poisoning		
							ARP poisoning and IP spoofing		

### ⚠ Disclaimer

The purpose of the Threat Matrix for Kubernetes is to educate readers on the potential of Kubernetes-based tactics, techniques, and procedures (TTPs). It is not to teach how to weaponize or specifically abuse them.

# Mitigations

#RSAC

Stronger Together

## Tactics

- Initial Access >
- Execution >
- Persistence >
- Privilege Escalation >
- Defense Evasion >
- Credential Access >
- Discovery >
- Lateral Movement >
- Collection >
- Impact >

## Tactics

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud resources	Images from a private registry	Data destruction
Compromised image In registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Collecting data from pod	Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Application credentials in configuration files	Exposed sensitive interfaces	Application credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container	Container service account			Access managed identity credentials	Instance Metadata API	Writable hostPath mount		
	Sidecar injection	Static pods			Malicious admission controller		CoreDNS poisoning		
							ARP poisoning and IP spoofing		

### ⚠ Disclaimer

The purpose of the Threat Matrix for Kubernetes is to educate readers on the potential of Kubernetes-based tactics, techniques, and procedures (TTPs). It is not to teach how to weaponize or specifically abuse them.

# Mitigations

#RSAC

Stronger Together

## Tactics

- Initial Access >
- Execution >
- Persistence >
- Privilege Escalation >
- Defense Evasion >
- Credential Access >
- Discovery >
- Lateral Movement >
- Collection >
- Impact >

## Tactics

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud resources	Images from a private registry	Data destruction
Compromised image In registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Collecting data from pod	Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Application credentials in configuration files	Exposed sensitive interfaces	Application credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container	Container service account			Access managed identity credentials	Instance Metadata API	Writable hostPath mount		
	Sidecar injection	Static pods			Malicious admission controller		CoreDNS poisoning		
							ARP poisoning and IP spoofing		

### ⚠ Disclaimer

The purpose of the Threat Matrix for Kubernetes is to educate readers on the potential of Kubernetes-based tactics, techniques, and procedures (TTPs). It is not to teach how to weaponize or specifically abuse them.

# Mitigations

#RSAC

Stronger Together

## Tactics

- Initial Access >
- Execution >
- Persistence >
- Privilege Escalation >
- Defense Evasion >
- Credential Access >
- Discovery >
- Lateral Movement >
- [Access cloud resources](#)
- Container service account
- Cluster internal networking
- Application credentials in configuration files
- Writable hostPath mount
- CoreDNS poisoning
- ARP poisoning and IP spoofing
- Collection >
- Impact >

## Access cloud resources

If the Kubernetes cluster is deployed in the cloud, in some cases attackers can leverage their access to a single container to get access to other cloud resources outside the cluster. For example, AKS uses several managed identities that are attached to the nodes, for the cluster operation. Similar identities exist also in EKS and GKE (EC2 roles and IAM service accounts, respectively). By default, running pods can retrieve the identities which in some configurations have privileged permissions. Therefore, if attackers gain access to a running pod in the cluster, they can leverage the identities to access external cloud resources.

Also, AKS has an option to authenticate with Azure using a service principal. When this option is enabled, each node stores service principal credentials that are located in `/etc/kubernetes/azure.json`. AKS uses this service principal to create and manage Azure resources that are needed for the cluster operation. By default, the service principal has contributor permissions in the cluster's Resource Group. Attackers who get access to this service principal file (by `hostPath` mount, for example) can use its credentials to access or modify the cloud resources.

## Mitigations

ID	Mitigation	Description
<a href="#">MS-M9003</a>	Adhere to least-privilege principle	Grant only necessary permission to the cloud identities.
<a href="#">MS-M9018</a>	Restrict the access of pods to IMDS	Restrict the access of pods to IMDS to restrict pods from getting access to cloud identities.
<a href="#">MS-M9019</a>	Allocate specific identities to pods	Use dedicated allocated identities to pods
<a href="#">MS-M9013</a>	Restrict over permissive containers	Block mounting volumes with access to cloud credentials.

**Info**

ID: MS-TA9020  
Tactic: [Privilege Escalation](#), [Lateral Movement](#)  
MITRE technique: [T1078.004](#)

# Mitigations

#RSAC

Stronger Together

## Tactics

- Initial Access >
- Execution >
- Persistence >
- Privilege Escalation >
- Defense Evasion >
- Credential Access >
- Discovery >
- Lateral Movement >
- [Access cloud resources](#)
- Container service account
- Cluster internal networking
- Application credentials in configuration files
- Writable hostPath mount
- CoreDNS poisoning
- ARP poisoning and IP spoofing
- Collection >
- Impact >

## Access cloud resources

If the Kubernetes cluster is deployed in the cloud, in some cases attackers can leverage their access to a single container to get access to other cloud resources outside the cluster. For example, AKS uses several managed identities that are attached to the nodes, for the cluster operation. Similar identities exist also in EKS and GKE (EC2 roles and IAM service accounts, respectively). By default, running pods can retrieve the identities which in some configurations have privileged permissions. Therefore, if attackers gain access to a running pod in the cluster, they can leverage the identities to access external cloud resources.

Also, AKS has an option to authenticate with Azure using a service principal. When this option is enabled, each node stores service principal credentials that are located in `/etc/kubernetes/azure.json`. AKS uses this service principal to create and manage Azure resources that are needed for the cluster operation. By default, the service principal has contributor permissions in the cluster's Resource Group. Attackers who get access to this service principal file (by `hostPath` mount, for example) can use its credentials to access or modify the cloud resources.

## Mitigations

ID	Mitigation	Description
<a href="#">MS-M9003</a>	Adhere to least-privilege principle	Grant only necessary permission to the cloud identities.
<a href="#">MS-M9018</a>	Restrict the access of pods to IMDS	Restrict the access of pods to IMDS to restrict pods from getting access to cloud identities.
<a href="#">MS-M9019</a>	Allocate specific identities to pods	Use dedicated allocated identities to pods
<a href="#">MS-M9013</a>	Restrict over permissive containers	Block mounting volumes with access to cloud credentials.

**Info**

ID: MS-TA9020  
Tactic: [Privilege Escalation](#), [Lateral Movement](#)  
MITRE technique: [T1078.004](#)

# Mitigations

#RSAC

Stronger Together

## Tactics

- Initial Access >
- Execution >
- Persistence >
- Privilege Escalation >
- Defense Evasion >
- Credential Access >
- Discovery >
- Lateral Movement >
- [Access cloud resources](#)
- Container service account
- Cluster internal networking
- Application credentials in configuration files
- Writable hostPath mount
- CoreDNS poisoning
- ARP poisoning and IP spoofing
- Collection >
- Impact >

## Access cloud resources

If the Kubernetes cluster is deployed in the cloud, in some cases attackers can leverage their access to a single container to get access to other cloud resources outside the cluster. For example, AKS uses several managed identities that are attached to the nodes, for the cluster operation. Similar identities exist also in EKS and GKE (EC2 roles and IAM service accounts, respectively). By default, running pods can retrieve the identities which in some configurations have privileged permissions. Therefore, if attackers gain access to a running pod in the cluster, they can leverage the identities to access external cloud resources.

Also, AKS has an option to authenticate with Azure using a service principal. When this option is enabled, each node stores service principal credentials that are located in `/etc/kubernetes/azure.json`. AKS uses this service principal to create and manage Azure resources that are needed for the cluster operation. By default, the service principal has contributor permissions in the cluster's Resource Group. Attackers who get access to this service principal file (by `hostPath` mount, for example) can use its credentials to access or modify the cloud resources.

## Mitigations

ID	Mitigation	Description
<a href="#">MS-M9003</a>	Adhere to least-privilege principle	Grant only necessary permission to the cloud identities.
<a href="#">MS-M9018</a>	Restrict the access of pods to IMDS	Restrict the access of pods to IMDS to restrict pods from getting access to cloud identities.
<a href="#">MS-M9019</a>	Allocate specific identities to pods	Use dedicated allocated identities to pods
<a href="#">MS-M9013</a>	Restrict over permissive containers	Block mounting volumes with access to cloud credentials.

**Info**

ID: MS-TA9020  
Tactic: [Privilege Escalation](#), [Lateral Movement](#)  
MITRE technique: [T1078.004](#)

# Mitigations

#RSAC

Stronger Together

## Mitigations

- Multi-factor authentication
- Restrict access to the API server using IP firewall
- Adhere to least-privilege principle
- Secure CI/CD environment
- Image assurance policy
- Enable Just In Time access to API server
- Network intrusion prevention
- Limit access to services over network
- Require strong authentication to services
- Restrict exec commands on pods
- Restrict container runtime using LSM
- Remove tools from container images
- Restrict over permissive containers
- Network segmentation
- Avoid running management interface on containers
- Restrict file and directory permissions
- Ensure that pods meet defined Pod Security Standards
- Restrict pod-to-pod access
- Allocate specific identities to pods

## Allocate specific identities to pods

When needed, allocate dedicated cloud identity per pod with minimal permissions, instead of inheriting the node's cloud identity. This prevents other pods from accessing cloud identities that are not necessary for their operation. The features that implement this separation are: Azure AD Pod Identity (AKS), Azure AD Workload identity (AKS), IRSA (EKS) and GCP Workload Identity (GCP).

### Techniques Addressed by Mitigation

ID	Name	Use
<a href="#">MS-TA9020</a>	Access cloud resources	Use dedicated allocated identities to pods
<a href="#">MS-TA9028</a>	Access Managed Identity credentials	Allocate specific identities to pods.

#### Info

ID: MS-M9019  
MITRE mitigation: -

# Mitigations

#RSAC

Stronger Together

## Tactics

- Initial Access >
- Execution >
- Persistence >
- Privilege Escalation >
- Defense Evasion >
- Credential Access >
- Discovery >
- Lateral Movement >
- Collection >
- Impact >

## Tactics

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud resources	Images from a private registry	Data destruction
Compromised image In registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Collecting data from pod	Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Application credentials in configuration files	Exposed sensitive interfaces	Application credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container	Container service account			Access managed identity credentials	Instance Metadata API	Writable hostPath mount		
	Sidecar injection	Static pods			Malicious admission controller		CoreDNS poisoning		
							ARP poisoning and IP spoofing		

### ⚠ Disclaimer

The purpose of the Threat Matrix for Kubernetes is to educate readers on the potential of Kubernetes-based tactics, techniques, and procedures (TTPs). It is not to teach how to weaponize or specifically abuse them.

# Mitigations

#RSAC

Stronger Together

## Tactics

- Initial Access >
- Execution >
- Persistence >
- Privilege Escalation >
- Defense Evasion >
- Credential Access >
- Discovery >
- Lateral Movement >
- Collection >
- Impact >

## Tactics

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud resources	Images from a private registry	Data destruction
Compromised image In registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Collecting data from pod	Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Application credentials in configuration files	Exposed sensitive interfaces	Application credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container	Container service account			Access managed identity credentials	Instance Metadata API	Writable hostPath mount		
	Sidecar injection	Static pods			Malicious admission controller		CoreDNS poisoning		
							ARP poisoning and IP spoofing		

### ⚠ Disclaimer

The purpose of the Threat Matrix for Kubernetes is to educate readers on the potential of Kubernetes-based tactics, techniques, and procedures (TTPs). It is not to teach how to weaponize or specifically abuse them.

# Mitigations

#RSAC

Stronger Together

## Tactics

- Initial Access >
- Execution >
- Persistence >
- Privilege Escalation >
- Defense Evasion >
- Credential Access >
- Discovery >
- Lateral Movement >
  - Access cloud resources
  - Container service account**
  - Cluster internal networking
  - Application credentials in configuration files
  - Writable hostPath mount
  - CoreDNS poisoning
  - ARP poisoning and IP spoofing
- Collection >
- Impact >

## Container service account

Service account (SA) represents an application identity in Kubernetes. By default, a Service Account access token is mounted to every created pod in the cluster and containers in the pod can send requests to the Kubernetes API server using the Service Account credentials. Attackers who get access to a pod can access the Service Account token (located in `/var/run/secrets/kubernetes.io/serviceaccount/token`) and perform actions in the cluster, according to the Service Account permissions. If RBAC is not enabled, the Service Account has unlimited permissions in the cluster. If RBAC is enabled, its permissions are determined by the RoleBindings \ ClusterRoleBindings that are associated with it.

An attacker which get access to the Service Account token can also authenticate and access the Kubernetes API server from outside the cluster and maintain access to the cluster.

## Mitigations

ID	Mitigation	Description
<a href="#">MS-M9025</a>	Disable Service Account Auto Mount	Disable service account auto mount.
<a href="#">MS-M9003</a>	Adhere to least-privilege principle	Configure the Kubernetes RBAC such that each service account will have the minimal necessary permissions for the application's functionality.

### Info

ID: MS-TA9016  
Tactic: [Credential Access](#), [Lateral Movement](#), [Persistence](#)  
MITRE technique: [T1528](#)

# Mitigations

#RSAC

Stronger Together

## Tactics

- Initial Access >
- Execution >
- Persistence >
- Privilege Escalation >
- Defense Evasion >
- Credential Access >
- Discovery >
- Lateral Movement >
  - Access cloud resources
  - [Container service account](#)
  - Cluster internal networking
  - Application credentials in configuration files
  - Writable hostPath mount
  - CoreDNS poisoning
  - ARP poisoning and IP spoofing
- Collection >
- Impact >

## Container service account

Service account (SA) represents an application identity in Kubernetes. By default, a Service Account access token is mounted to every created pod in the cluster and containers in the pod can send requests to the Kubernetes API server using the Service Account credentials. Attackers who get access to a pod can access the Service Account token (located in `/var/run/secrets/kubernetes.io/serviceaccount/token`) and perform actions in the cluster, according to the Service Account permissions. If RBAC is not enabled, the Service Account has unlimited permissions in the cluster. If RBAC is enabled, its permissions are determined by the RoleBindings \ ClusterRoleBindings that are associated with it.

An attacker which get access to the Service Account token can also authenticate and access the Kubernetes API server from outside the cluster and maintain access to the cluster.

**Info**

ID: MS-TA9016  
Tactic: [Credential Access](#), [Lateral Movement](#), [Persistence](#)  
MITRE technique: [T1528](#)

## Mitigations

ID	Mitigation	Description
<a href="#">MS-M9025</a>	Disable Service Account Auto Mount	Disable service account auto mount.
<a href="#">MS-M9003</a>	Adhere to least-privilege principle	Configure the Kubernetes RBAC such that each service account will have the minimal necessary permissions for the application's functionality.

# Mitigations

#RSAC

Stronger Together

## Mitigations

- Multi-factor authentication
- Restrict access to the API server using IP firewall
- Adhere to least-privilege principle
- Secure CI/CD environment
- Image assurance policy >
- Enable Just In Time access to API server
- Network intrusion prevention
- Limit access to services over network
- Require strong authentication to services
- Restrict exec commands on pods
- Restrict container runtime using LSM
- Remove tools from container images
- Restrict over permissive containers
- Network segmentation
- Avoid running management interface on containers
- Restrict file and directory permissions
- Ensure that pods meet defined Pod Security Standards
- Restricting cloud metadata API access
- Allocate specific identities to

## Disable service account auto mount

By default, a service account is mounted to every pod. If the application doesn't require access to the Kubernetes API, disable the service account auto-mount by specifying `automountServiceAccountToken: false` in the pod configuration.

### Techniques Addressed by Mitigation

ID	Name	Use
<a href="#">MS-TA9016</a>	Container service account	Disable service account auto mount.

#### Info

ID: MS-M9025  
MITRE mitigation: -

**RSA**Conference™2023



**Stronger  
Together**

## Key takeaways

# Key takeaways



- Implement a holistic strategy for K8s security by considering both the cluster and cloud levels.
- Identities are a key aspect of K8s security: Monitor their activity using auditing tools.
- Adhere to the least-privilege principle.
- Use mitigation measures to prevent potential attacks.

**RSA**Conference™2023



**Stronger  
Together**

**Thank you!**