dtdg.co/kubecon23-managed-k8s-security

Mind The Gap! Bringing Together Cloud Services and Managed K8s Environments

Christophe Tafani-Dereeper, Diego Comas

v1.SelfSubjectAccessReview

Christophe Tafani-Dereeper
Cloud Security Researcher & Advocate

DATADOG

Diego Comas
Head of Security

sourcegraph

Today's agenda

Common security pitfalls in managed Kubernetes environments

AWS
Elastic Kubernetes Service

GCP
Google Kubernetes Engine

Azure
Azure Kubernetes Service

Teaser

Follow the journey!

Kate
Cloud-Native Software Engineer

The Cat Classifier app

As a: Cat enthusiast

I want: to be able to easily identify the breed of any cat picture

So that: I can find the cat of my dreams and feel fulfilled in life

Deploying the app to an EKS cluster

AWS

EKS cluster

Deploying the app to an EKS cluster

```
$ aws eks update-kubeconfig --name eks-cluster

$ kubectl get nodes
error: You must be logged in to the server (Unauthorized)
```

Deploying an EKS cluster


```
time="2023-04-08T08:31:25Z" level=warning
method=POST path=/authenticate
msg="access denied" error="ARN is not mapped"
```

What happened?

- AWS account admin != K8s cluster admin
- Permissions are managed inside the cluster
- Whoever created the EKS cluster has **invisible**, **immutable** admin rights

> When you create an Amazon EKS cluster, the IAM entity user or role, such as a federated user that creates the cluster, is automatically granted `system:masters` permissions in the cluster's RBAC configuration. This access cannot be removed and is not managed through the `aws-auth` ConfigMap. Therefore it is a good idea to create the cluster with a dedicated IAM role and regularly audit who can assume this role. This role should not be used to perform routine

https://aws.github.io/aws-eks-best-practices/security/docs/iam/#create-the-cluster-with-a-dedicated-iam-role

How EKS authenticates humans

`aws-auth` ConfigMap in the `kube-system` namespace

```
apiVersion: v1
kind: ConfigMap
data:
 mapRoles: |
   - rolearn: arn:aws:iam::012345678901:role/eks-cluster-NodeInstanceRole
     groups: [system:bootstrappers, system:nodes]
     username: system:node:{{EC2PrivateDNSName}}
```

https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html

Implications for incident response

- Make sure you have access to your clusters *ahead of time*
- Explicitly define your permissions
- Hard to answer "who has admin access to my cluster"
  - Need to correlate with CloudTrail event **`eks:CreateCluster`**

```yaml
apiVersion: v1
kind: ConfigMap
data:
 mapRoles: |
   - rolearn: arn:aws:iam::012345678901:role/eks-cluster-NodeInstanceRole
     groups: [system:bootstrappers, system:nodes]
     username: system:node:{{EC2PrivateDNSName}}

   - rolearn: arn:aws:iam::012345678901:role/account-admin
     groups: [system:masters]
     username: admin
```

https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html

Kubeconfig to access an EKS cluster

Kubeconfig generated by "`aws eks update-kubeconfig`"

```
users:
- name: arn:aws:eks:us-east-1:012345678901:cluster/eks-cluster
  user:
   exec:
     apiVersion: client.authentication.k8s.io/v1beta1
     command: aws
     args:
     - --region
     - us-east-1
     - eks
     - get-token
     - --cluster-name
     - eks-cluster
```

```
aws eks get-token --cluster-name eks-cluster
```

```
https://sts.us-east-1.amazonaws.com/?Action=GetCallerIdentity&Version=2011-06-15&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=ASIAZ3MSJV4W72QRIR5R%2F20230411%2Fus-east-1%2Fsts%2Faws4_request&X-Amz-Date=20230411T085410Z&X-Amz-Expires=60&X-Amz-SignedHeaders=host%3Bx-k8s-aws-id&X-Amz-Security-Token=IQoJb3JpZ2luX2VjEL
```

Pre-signed AWS API call for `sts:GetCallerIdentity`

```
aws eks get-token --cluster-name eks-cluster  | jq -r .status.token | cut -d. -f2 | base64 -d
```

Generate pre-signed URL
for sts:GetCallerIdentity

1

`Authorization: Bearer <pre-signed URL>`

EKS managed control plane

API Server

Webhook authenticator

4

Kate is authenticated!

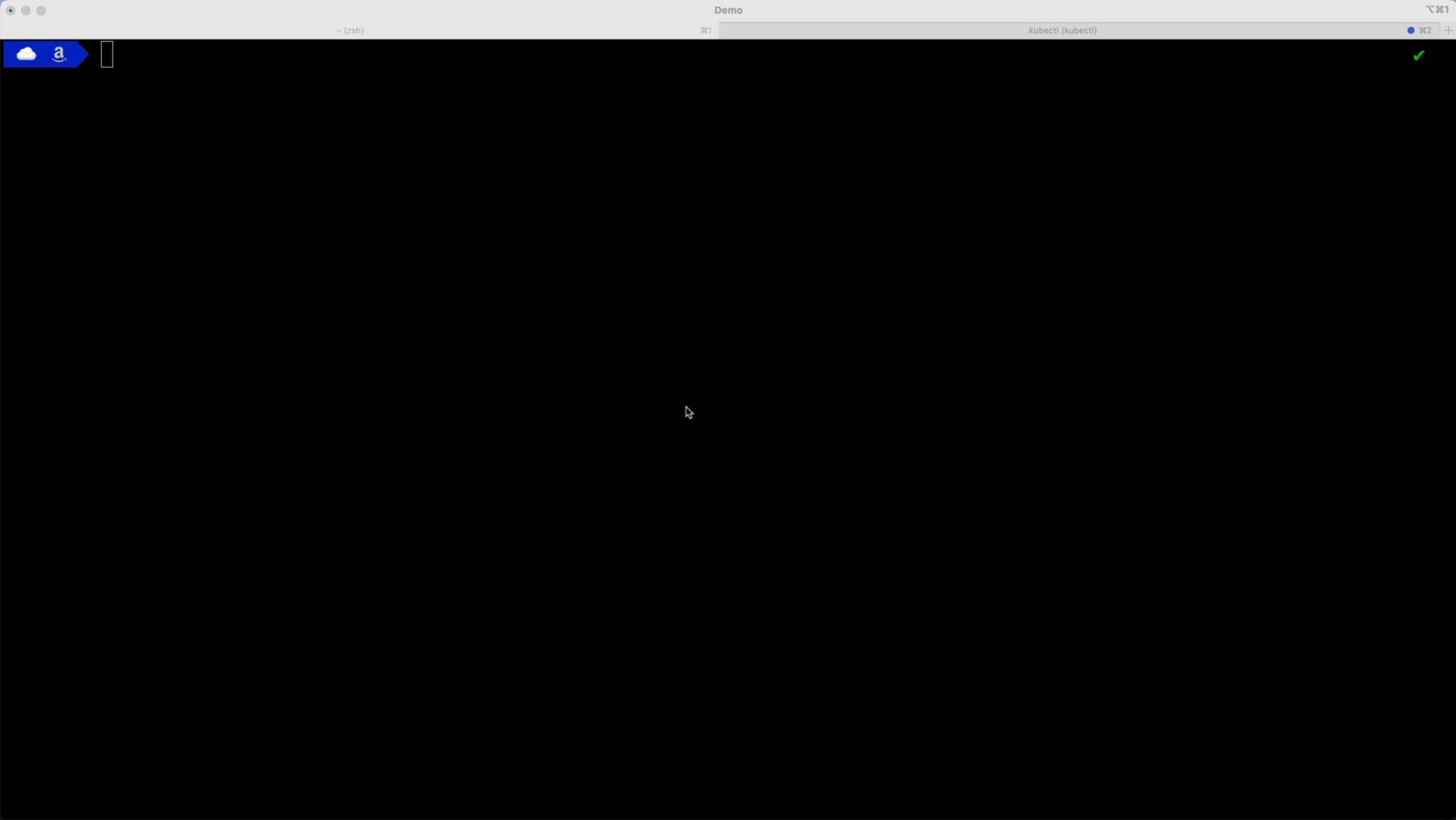
2

```
"GetCallerIdentityResult": {
     "Account": "012345678901",
     "Arn":"arn:aws:sts::012345678901:assumed-role/admin",
     "UserId": "AROAXXX:admin"
}
```

3

AWS IAM

https://github.com/kubernetes-sigs/aws-iam-authenticator/blob/f131685174cf16630a49baef08c472f99db47f70/pkg/token/token.go#L465


Demo
~ (zsh)
kubectl (kubectl)

In other clouds

Azure

- Permissions managed through Azure or in-cluster through (Cluster)RoleBindings

Azure Kubernetes Service Cluster Admin Role	List cluster admin credential action.
Azure Kubernetes Service Cluster User Role	List cluster user credential action.
Azure Kubernetes Service Contributor Role	Grants access to read and write Azure Kubernetes Service clusters
Azure Kubernetes Service RBAC Admin	Lets you manage all resources un cluster/namespace, except update resource quotas and namespaces
Azure Kubernetes Service RBAC Cluster Admin	Lets you manage all resources in

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: pod-reader
subjects:
- kind: Group
  name: [Azure AD groupObjectId]
```

https://learn.microsoft.com/en-us/azure/aks/azure-ad-rbac?tabs=portal

In other clouds

Google Cloud

➢ Permissions managed through GCP IAM **or** in-cluster (Cluster)RoleBinding

Kubernetes Engine roles
Kubernetes Engine Cluster Admin (`roles/container.clusterAdmin`)
Kubernetes Engine Cluster Viewer (`roles/container.clusterViewer`)

Lowest-level resources where you can grant this role:

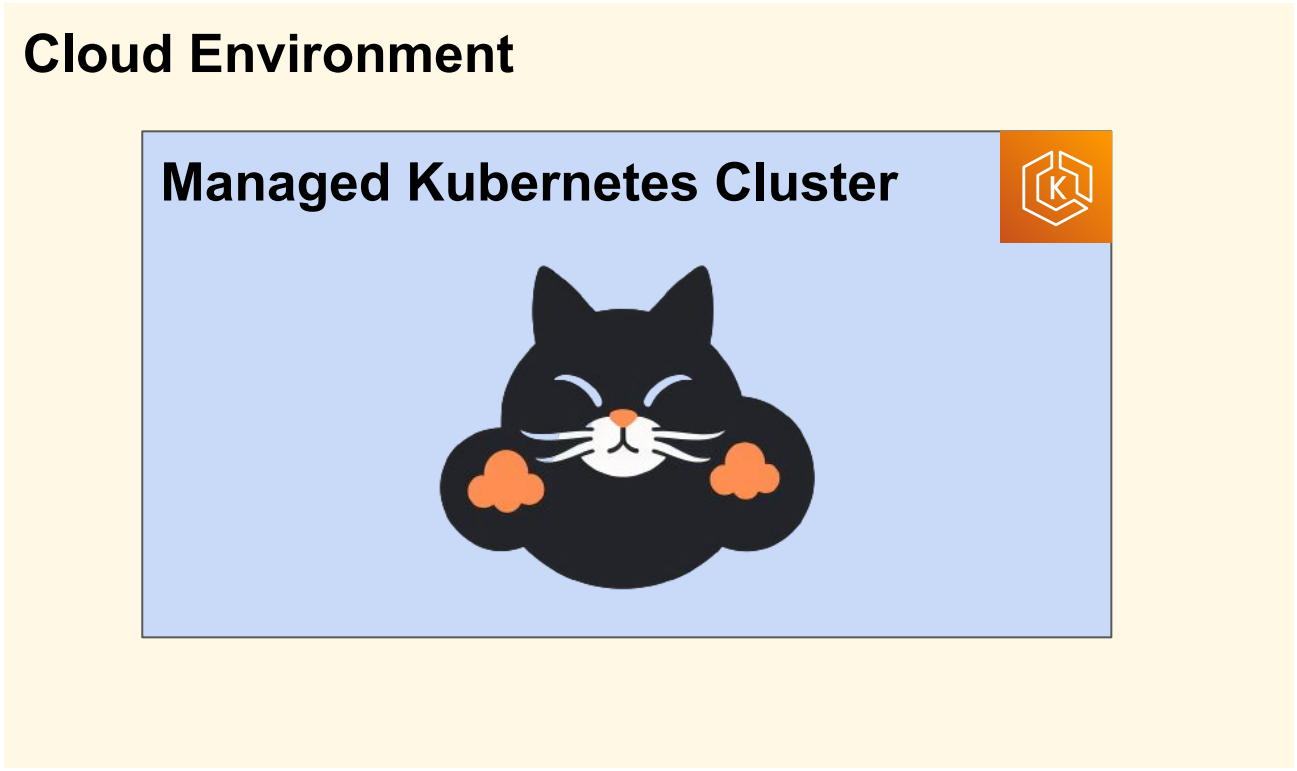
- Project

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: pod-reader
subjects:
- kind: User
 name: christophe@somewhereinthe.cloud
- kind: Group
  name: k8s-readers@somewhereinthe.cloud
```

Summary

	AWS EKS	Azure AKS	GCP GKE
Permissions managed in-cluster	✅	✅	✅
Permissions managed through native IAM	❌	✅	✅

Deploying the Cat Classifier application



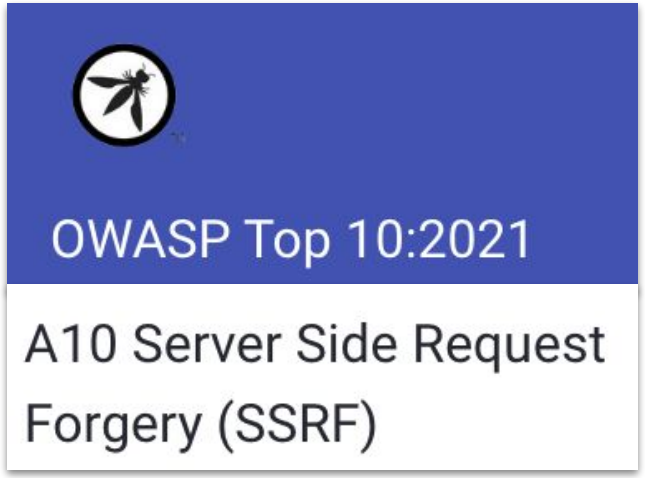
Adding an image import feature

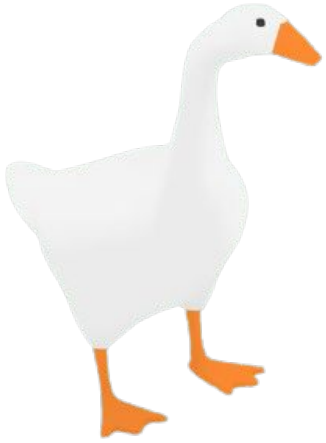
```
requests.get(req.args.get('url'))
```

i.imgur.com/cat.png

Cat Classifier

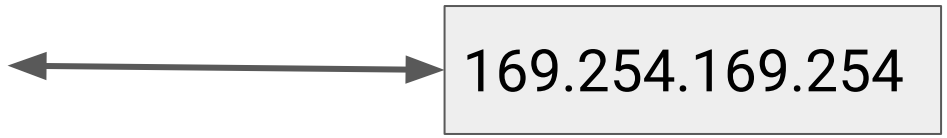
i.imgur.com





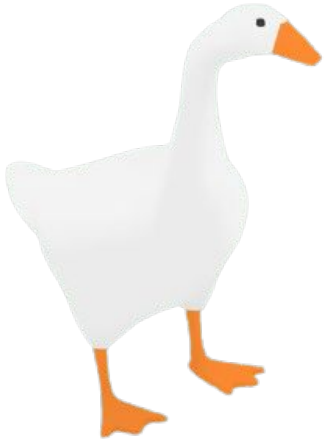
169.254.169.254

Cat Classifier



Instance Metadata Service

169.254.169.254/latest/meta-data/iam
/**security-credentials**/NodeInstanceRole



169.254.169.254

```
{
  "Code": "Success",
  "AccessKeyId": "ASIAZ3MSJV4WZZ2DUXOB",
  "SecretAccessKey": "mTxQZbVCCyOiYWIdq0Y40jrP0Leu3",
  "SessionToken": "IQoJb3JpZ2luX2VjEL//...",
  "Expiration": "..."
}
```

Instance Metadata Service

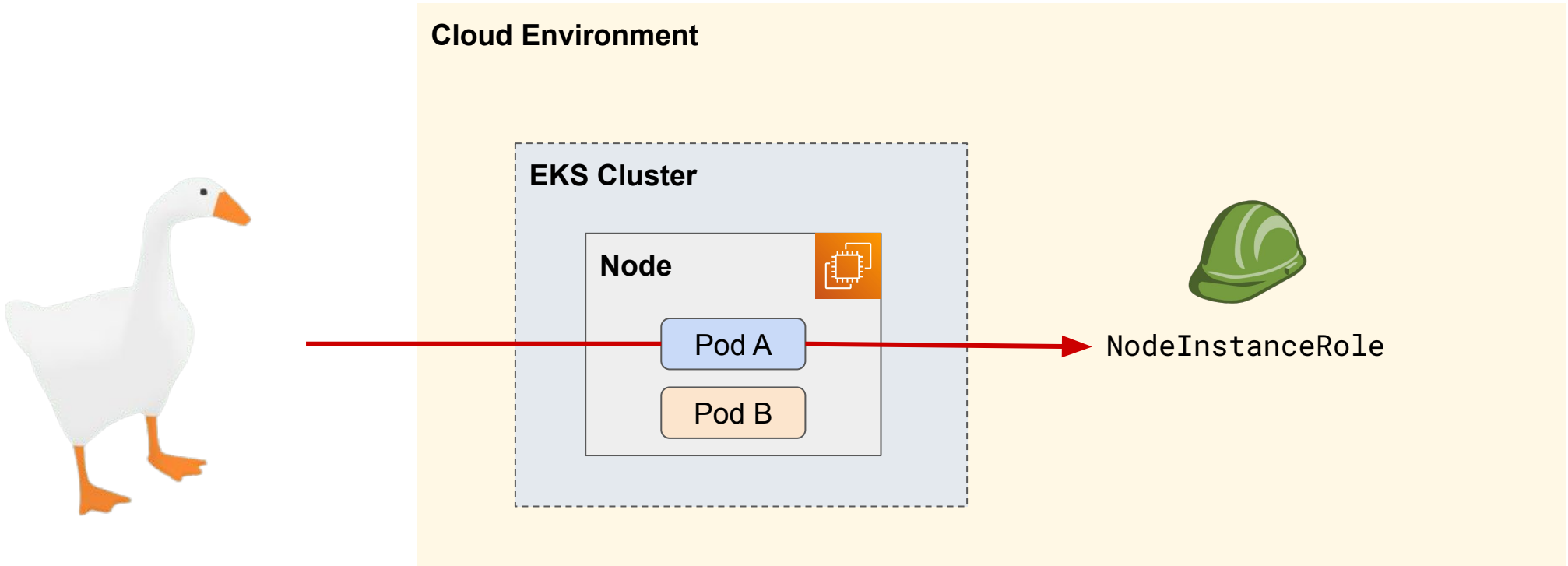
Server-Side Request Forgery to the IMDS

- One of the most common causes for public cloud breaches

- Juicy target for attackers:
  - Steal worker node cloud credentials
  - Authenticate to the cluster as a worker node
  - ⇒ impersonate any pod running on the node

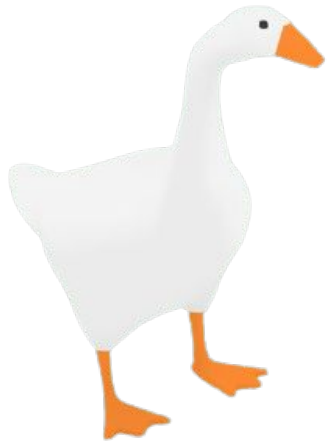
https://blog.christophetd.fr/cloud-security-breaches-and-vulnerabilities-2021-in-review/
https://securitylabs.datadoghq.com/articles/public-cloud-breaches-2022-mccarthy-hopkins/
https://blog.calif.io/p/privilege-escalation-in-eks

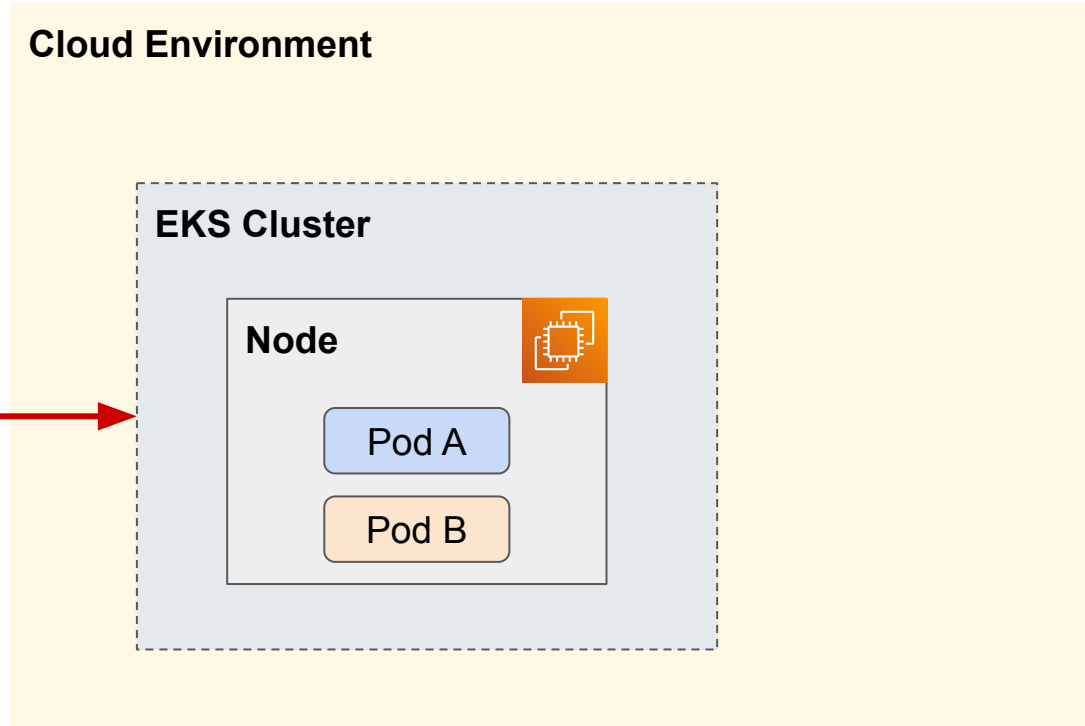
Server-Side Request Forgery to the IMDS

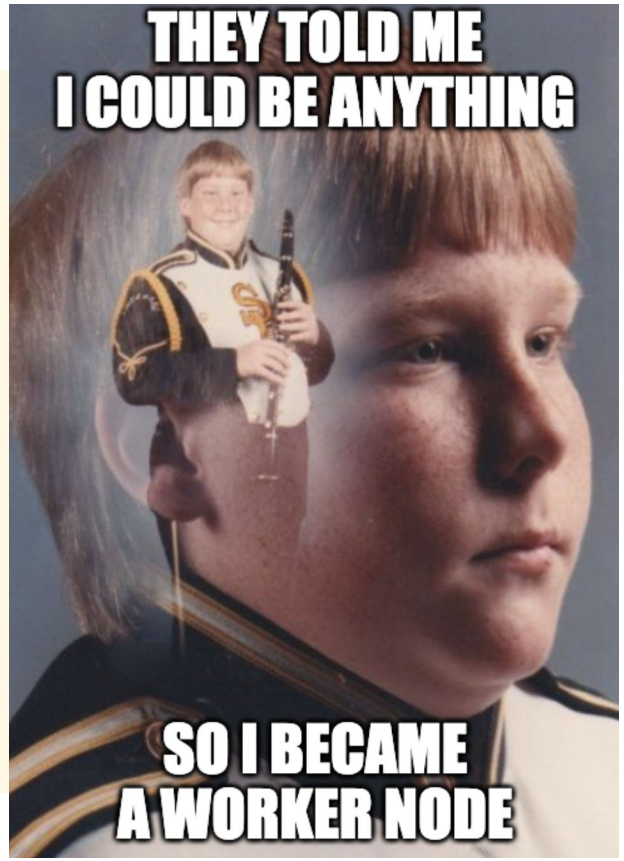


https://blog.christophetd.fr/privilege-escalation-in-aws-elastic-kubernetes-service-eks-by-compromising-the-instance-role-of-worker-nodes/

Server-Side Request Forgery to the IMDS

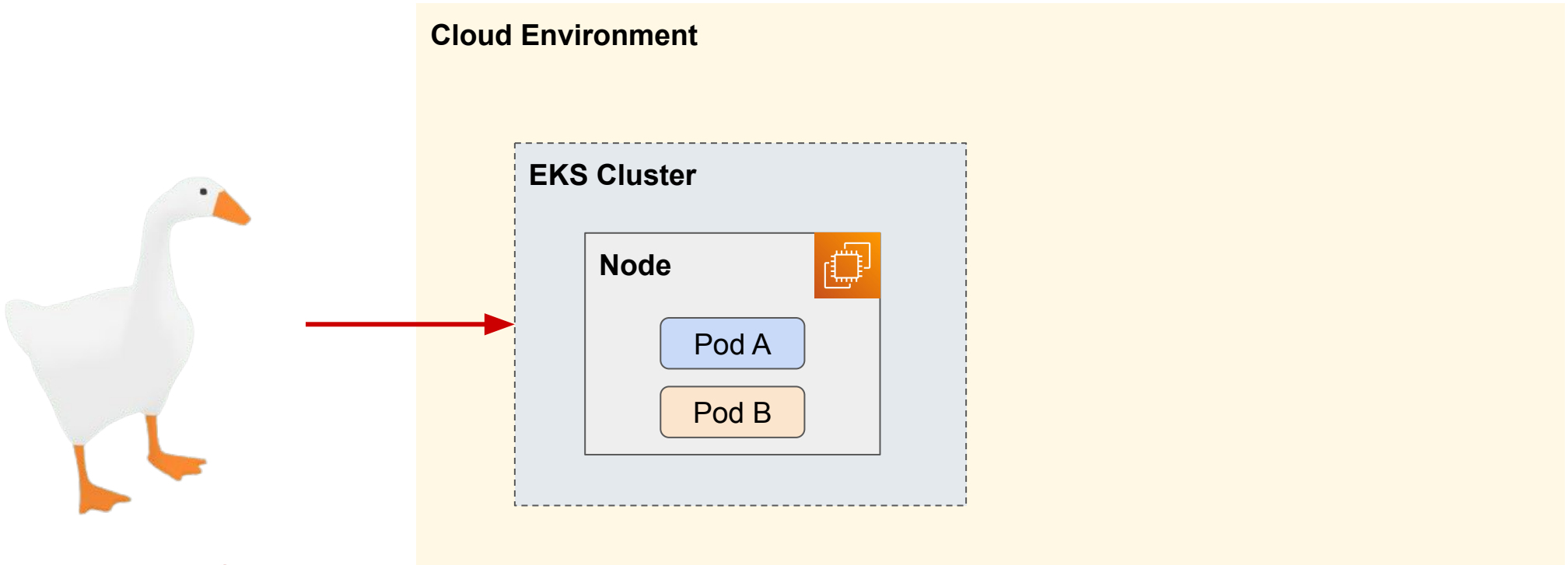






Authenticated as system:nodes

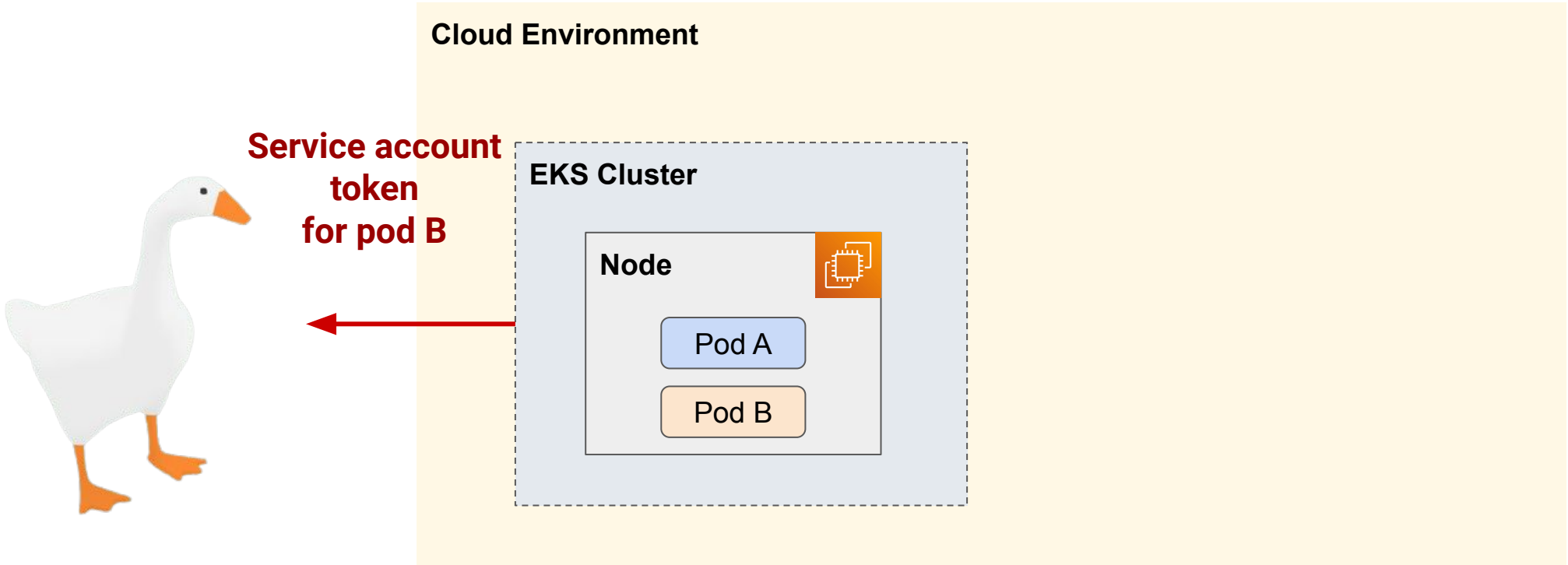
Server-Side Request Forgery to the IMDS



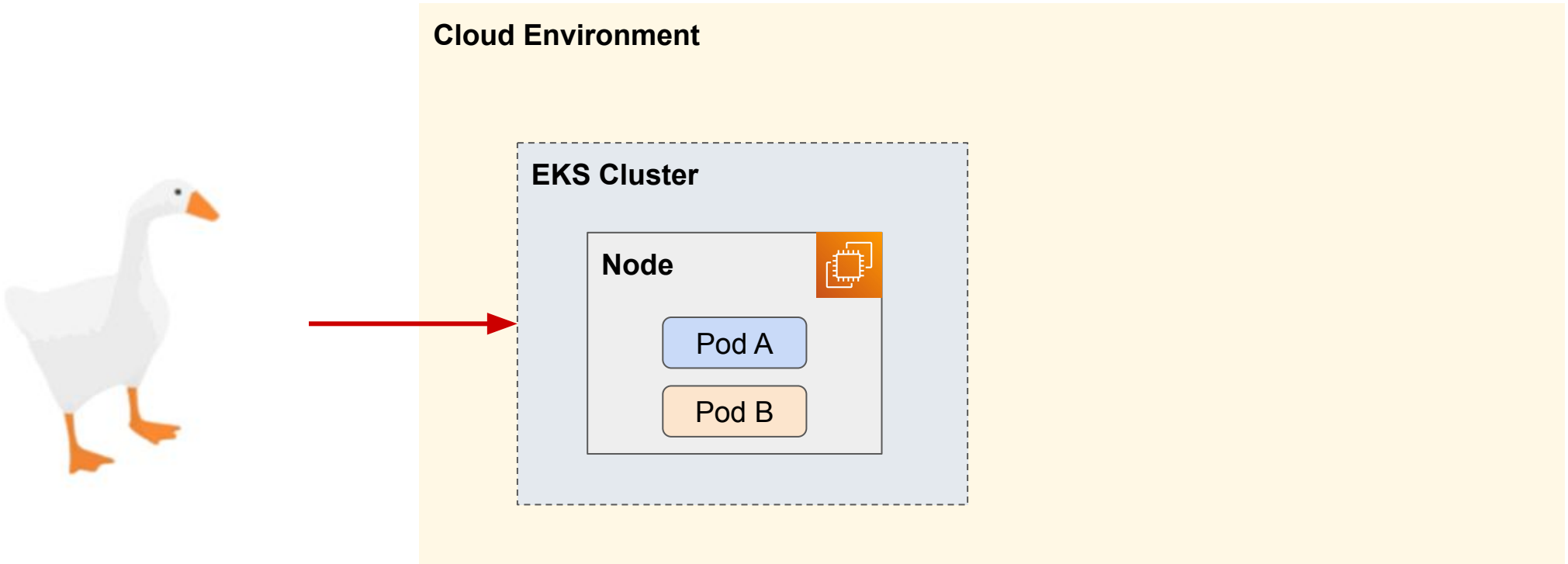
Create token for pod B

https://blog.calif.io/p/privilege-escalation-in-eks

Server-Side Request Forgery to the IMDS



Server-Side Request Forgery to the IMDS



Authenticated as pod-b

Demo: SSRF to the IMDS

```
    "UserId": "[illegible]",
    "Account": "[illegible]",
    "Arn": "arn:aws:sts::[illegible]:assumed-role/eksctl-[illegible]-cluster-nodegroup-ng-[illegible]-NodeInstanceRole-[illegible]/i-[illegible]"
}
kubectl-whoami
system:node:ip-192-168-[illegible].eu-west-1.compute.internal
kubectl auth can-i --list
Warning: the list may be incomplete: node authorizer does not support user rule resolution
Resources                                                        Non-Resource URLs   Resource Names     Verbs
selfsubjectaccessreviews.authorization.k8s.io                    []                  []                 [create]
selfsubjectrulesreviews.authorization.k8s.io                     []                  []                 [create]
certificatesigningrequests.certificates.k8s.io/selfnodeserver    []                  []                 [create]
                                                                 [/api/*]            []                 [get]
                                                                 [/api]              []                 [get]
                                                                 [/apis/*]           []                 [get]
                                                                 [/apis]             []                 [get]
                                                                 [/healthz]          []                 [get]
                                                                 [/healthz]          []                 [get]
                                                                 [/livez]            []                 [get]
                                                                 [/livez]            []                 [get]
                                                                 [/openapi/*]        []                 [get]
                                                                 [/openapi]          []                 [get]
                                                                 [/readyz]           []                 [get]
                                                                 [/readyz]           []                 [get]
                                                                 [/version/]         []                 [get]
                                                                 [/version/]         []                 [get]
                                                                 [/version]          []                 [get]
                                                                 [/version]          []                 [get]
podsecuritypolicies.policy                                       []                  [eks.privileged]   [use]
kubectl auth can-i --list
```

Demo summary

1. Compromised node AWS IAM credentials through the instance metadata service

2. Authenticated to the AWS API as the worker node

3. Created a service account token for another pod running on the node
    - Cluster admin rights!

Impact & exploitability of SSRFs

- Node IAM role has limited but still powerful permissions
  - List/describe EKS clusters, EC2 instances, VPCs
  - Pull all container images in the account
  - List and nuke all network interfaces in the account

- Exploitable when IMDSv2 is not enforced (default)

Impact & exploitability of SSRFs

- Nodes don't have permissions by default
  - iteratively added, e.g. when enabling the ACR integration

- Access to the IMDS requires a specific `Metadata: True` header

Impact & exploitability of SSRFs

Google Cloud

- Nodes use the default compute service account with limited scope
  - still allows to **list and read from all GCS buckets BigQuery datasets** in the project

When you create a new Compute Engine instance, it is automatically configured with the following access scopes:

- Read-only access to Cloud Storage:
  `https://www.googleapis.com/auth/devstorage.read_only`

BigQuery API, v2

Scopes	
https://www.googleapis.com/auth/devstorage.read_only	View your data in Google Cloud Storage

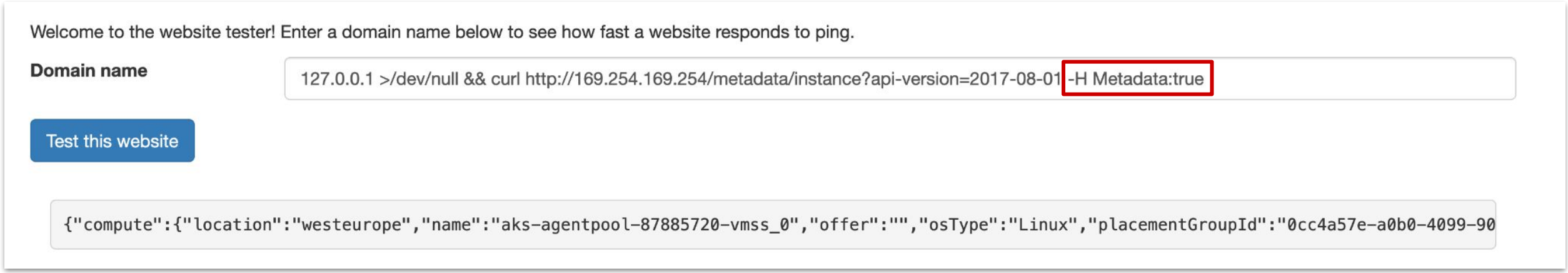
Cloud Storage JSON API, v1

Scopes	
https://www.googleapis.com/auth/devstorage.read_only	View your data in Google Cloud Storage

- Access to the IMDS requires a specific `Metadata-Flavor: Google` header

These countermeasures aren't solving credentials theft!

- They make credentials harder to steal **through SSRF**
- … but **none of this solves the issue of a pod is fully compromised**
  - command injection, SQL injection, remote code execution



Attackers love credentials

```
curl -XPOST -k https://$KT:10250/run/$NS/$PN/$CN" \
 -d cmd="wget -q -O /tmp/.instance http://169.254.169.254/latest/meta-data/iam/security-credentials/"
```

https://attack.mitre.org/groups/G0139/
https://www.mandiant.com/resources/blog/cloud-metadata-abuse-unc2903

Attackers love credentials

- Harvest AWS creds from metadata service

```
get_aws_data(){

AWS_INFO=$(dload http://169.254.169.254/latest/meta-data/iam/info | tr '\0' '\n')
AWS_1_EC2=$(dload http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance | tr '
AWS_1_IAM_NAME=$(dload http://169.254.169.254/latest/meta-data/iam/security-credentials/)
```

https://permiso.io/blog/s/christmas-cloud-cred-harvesting-campaign/

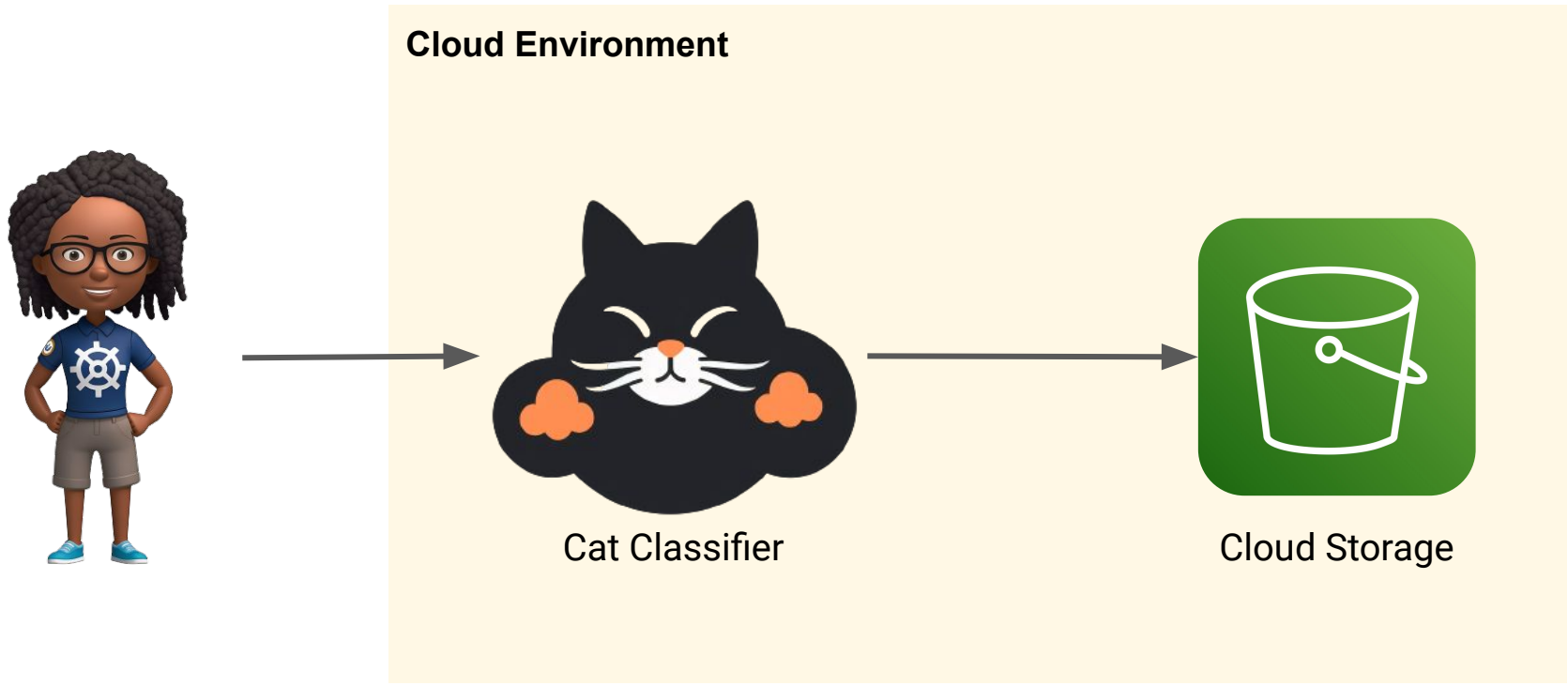
Mitigating pods stealing node credentials

- GCP: Enable GKE Workload Identity
  - Enabled by default in Autopilot clusters
  - "*will be enabled by default in a future release*"

- AWS / Azure: Block access to IMDS with a `NetworkPolicy`

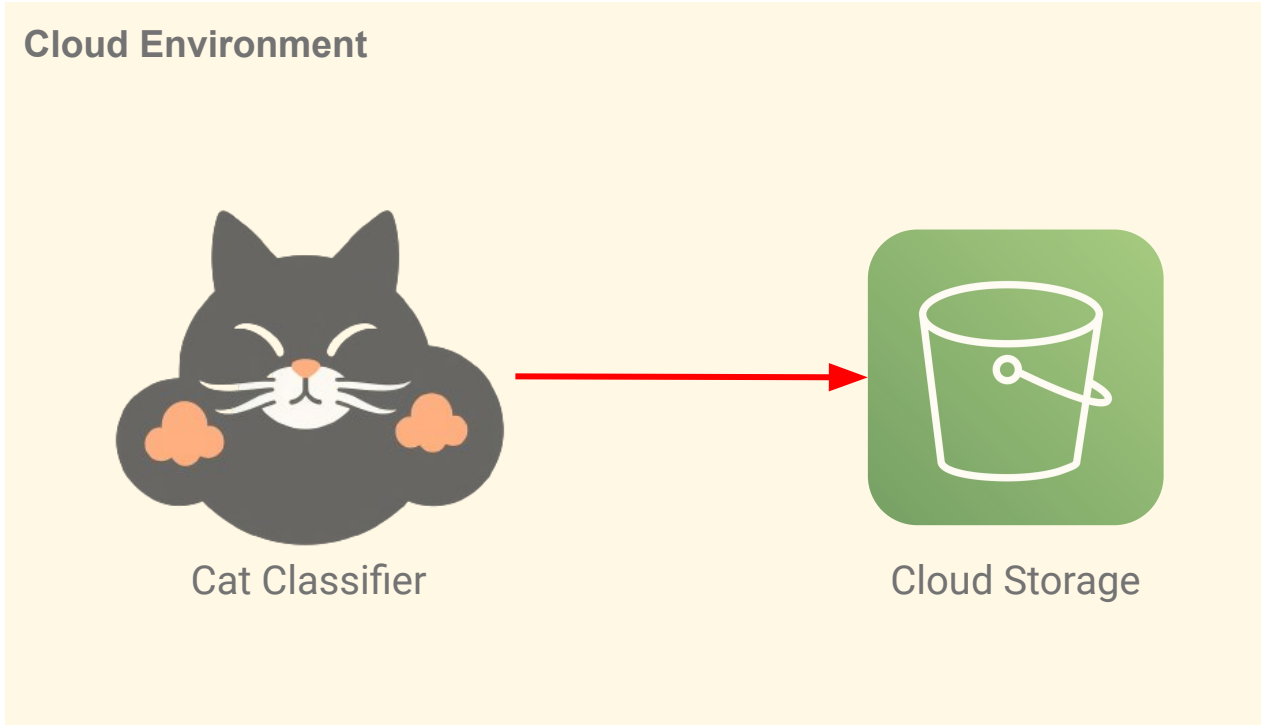
https://aws.github.io/aws-eks-best-practices/security/docs/iam/#restrict-access-to-the-instance-profile-assigned-to-the-worker-node
https://cloud.google.com/kubernetes-engine/docs/how-to/protecting-cluster-metadata

Back to the Cat Classifier application



Back to the Cat Classifier application

How do I authenticate my application against cloud provider APIs?



GPT-3.5

How do I give my Kubernetes pod access to upload files to an S3 bucket?

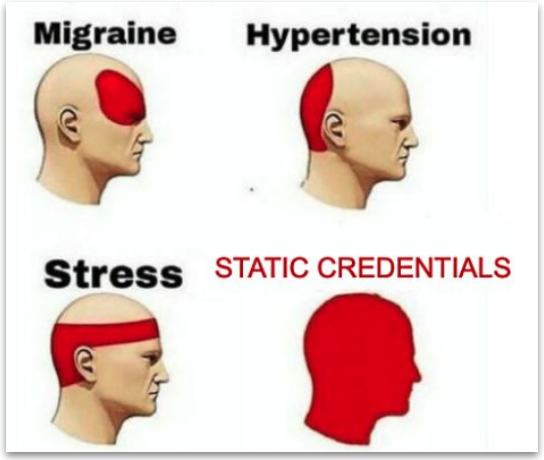
To give your Kubernetes pod access to upload files to an S3 bucket, you can follow these steps:

1. Create an IAM user in AWS that has S3 write access.

4. Create a Kubernetes secret that contains your AWS access key and secret key:

```
apiVersion: v1
kind: Secret
metadata:
  name: aws-secret
type: Opaque
data:
  AWS_ACCESS_KEY_ID: <base64-encoded-access-key>
  AWS_SECRET_ACCESS_KEY: <base64-encoded-secret-key>
```

Avoid static, long-lived credentials




```
186     <string name="all_scoresBtnTxt">All Scores</string>
187     <string name="amezon_app_id">AKIA[redacted]JA</string>
188     <string name="amezon_app_key">Yxwz[redacted]rLg5W</string>
```

2 lines (2 sloc) | 105 Bytes

```
1  aws_access_key_id = AK[redacted]DA
2  aws_secret_access_key = Cu[redacted]j1
```

Extra {"aws_access_key_id":"AK[redacted]", "aws_secret_access_key":"[redacted]"}

Avoid static, long-lived credentials

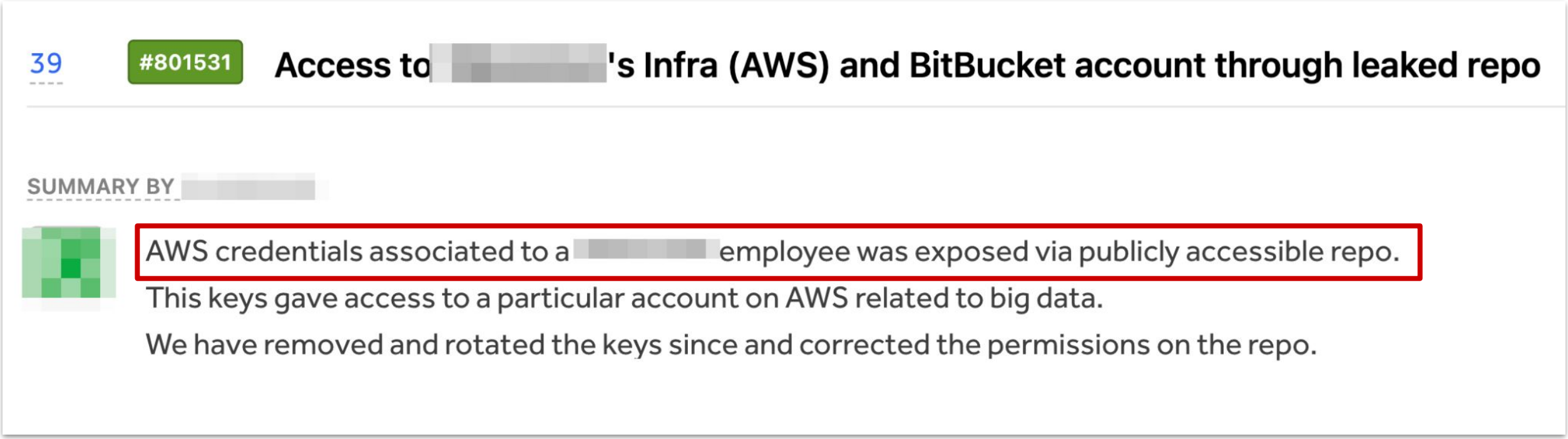
npm security update: Attack campaign using stolen OAuth tokens

Using their initial foothold of OAuth user tokens for GitHub.com, the actor was able to exfiltrate a set of private npm repositories, some of which included secrets such as AWS access keys.

Using one of these AWS access keys, the actor was able to gain access to npm's AWS infrastructure.

https://github.blog/2022-05-26-npm-security-update-oauth-tokens/

Avoid static, long-lived credentials



https://hackerone.com/reports/801531

Authenticating a K8s application to cloud provider APIs

Use a **platform-managed identity** instead

- AWS IAM Roles for Service Accounts
- GKE Workload Identity
- ~~Azure AD Pod Identity (Preview)~~ - *deprecated as of October 2022*
- Azure AD Workload Identity (Preview)

https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html
https://learn.microsoft.com/en-us/azure/aks/workload-identity-overview
https://cloud.google.com/kubernetes-engine/docs/concepts/workload-identity

IAM Roles for Service Accounts in AWS

Create an IAM role with an appropriate trust policy

```
{
 "Effect": "Allow",
 "Principal": {
   "Federated": "arn:aws:iam::012345678901:oidc-provider/oidc.eks.us-east-1.amazonaws.com/id/OIDC-PROVIDER-ID"
 },
 "Action": "sts:AssumeRoleWithWebIdentity",
 "Condition": {
   "StringEquals": {
     "oidc.eks.us-east-1.amazonaws.com/id/OIDC-PROVIDER-ID:aud": "sts.amazonaws.com",
     "oidc.eks.us-east-1.amazonaws.com/id/OIDC-PROVIDER-ID:sub": "system:serviceaccount:default:cat-classifier"
   }
 }
}
```

IAM Roles for Service Accounts in AWS

Create a K8s service account

```yaml
apiVersion: v1
kind: ServiceAccount
name: cat-classifier
metadata:
 annotations:
   eks.amazonaws.com/role-arn: arn:aws:iam::012345678901:role/cat-classifier
```

IAM Roles for Service Accounts in AWS

Deploy a pod running under this service account

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: cat-classifier
spec:
 serviceAccountName: cat-classifier
 containers:
 - image: ghcr.io/kate/cat-classifier
   name: app
```

JWT injected automatically in the pod

```
pod$ cat /var/run/secrets/eks.amazonaws.com/serviceaccount/token
```


```
{
 "aud": ["sts.amazonaws.com"],
 "iss": "https://oidc.eks.us-east-1.amazonaws.com/id/OIDC-PROVIDER-ID",
 "sub": "system:serviceaccount:default:cat-classifier",
 "kubernetes.io": {
   "namespace": "default",
   "pod": {
     "name": "cat-classifier"
   },
   "serviceaccount": {
     "name": "cat-classifier"
   }
 }
}
```

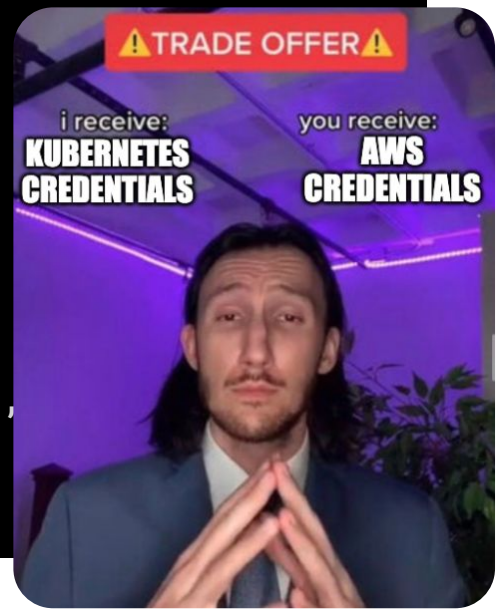
It's all transparent to the application!

Automatically picked up by the AWS CLI / SDKs

```
pod# aws sts get-caller-identity
{
    "UserId": "..",
    "Account": "012345678901",
    "Arn": "arn:aws:sts::012345678901:assumed-role/cat-classifier/botocore-session"
}
```

Exchanging a K8s service account token for cloud credentials

```
pod# aws sts assume-role-with-web-identity \
  --web-identity-token file:///var/run/secrets/eks.amazonaws.com/serviceaccount/token \
  --role-arn arn:aws:iam::012345678901:role/cat-classifier
{
 "Credentials": {
     "AccessKeyId": "ASIAZ3MSJV4WZZ2DUXOB",
     "SecretAccessKey": "mTxQZbVCCyOiYWIdq0Y40jrP0Leu3",
     "SessionToken": "IQoJb3JpZ2luX2VjEL//...",
     "Expiration": "..."
 },
 "SubjectFromWebIdentityToken": "system:serviceaccount:default:cat-classifier",
 "Audience": "sts.amazonaws.com"
}
```



GPT-3.5

How do I give my Kubernetes pod access to upload files to an S3 bucket?

To give your Kubernetes pod access to upload files to an S3 bucket, you can follow these steps:

1. Create an IAM user in AWS that has S3 write access.

Any idea to make it more secure?

To make the solution even more secure, you can consider the following additional measures:

1. Use AWS IAM Roles for Service Accounts (IRSA) instead of AWS IAM User credentials. IRSA allows you to provide an IAM role to a Kubernetes ServiceAccount. This way, you can avoid managing AWS access keys and secrets for each pod individually.

Why didn't you suggest the secure solution by default?

As an AI language model, I don't make assumptions about the security requirements of different scenarios,

What about other cloud providers?

	AWS	GCP	Azure
Mechanism name	"IAM Roles for Service Accounts"	"GKE Workload Identity"	"Azure AD Workload Identity"
"Unit of identity"	IAM Roles	Service Accounts	Managed Identity
How applications retrieve credentials inside the pod	**Service account token with custom audience**	**Instance Metadata Service (intercepted by a DaemonSet)**	**Service account token with custom audience***
Mechanism to retrieve credentials under the hood	STS AssumeRole	Service account impersonation	Federated identity credentials

https://azure.github.io/azure-workload-identity/docs/introduction.html

* *Optionally, Instance Metadata API intercepted by a sidecar container that can be automatically injected*

In other clouds

- Azure: Similar to AWS IAM Roles for Service Accounts
- Service account token (JWT) with custom audience

```
pod$ cat /var/run/secrets/azure/tokens/azure-identity-token
```

```
{
 "aud": ["api://AzureADTokenExchange"],
 "iss": "https://westeurope.oic.prod-aks.azure.com/<uuid>/<uuid>/",
 "sub": "system:serviceaccount:default:cat-classifier",
 ...
}
```

https://azure.github.io/azure-workload-identity/docs/introduction.html

Optionally, Instance Metadata API intercepted by a sidecar container that can be automatically injected

In other clouds

Google Cloud

- DaemonSet intercepting pod requests to the IMDS

```
kubectl get pods -n kube-system
NAME                                                  READY   STATUS
event-exporter-gke-857959888b-6zbsw                   2/2     Running
fluentbit-gke-f8h4t                                   2/2     Running
fluentbit-gke-tmlxc                                   2/2     Running
fluentbit-gke-zdgbl                                   2/2     Running
gke-metadata-server-24jkl                             1/1     Running
gke-metadata-server-c2jd8                             1/1     Running
gke-metadata-server-qbjqs                             1/1     Running
```

```
gke-cluster-1-default-pool-0af84212-mk4x / # iptables-save | grep metadata
-A PREROUTING -d 169.254.169.254/32 ! -i eth0 -p tcp -m tcp --dport 80 -m comment --comment "metadata-concealment: bridge traffic t
adata server goes to metadata proxy" -j DNAT --to-destination 169.254.169.252:988
```

Preventing worker node credentials theft from pods

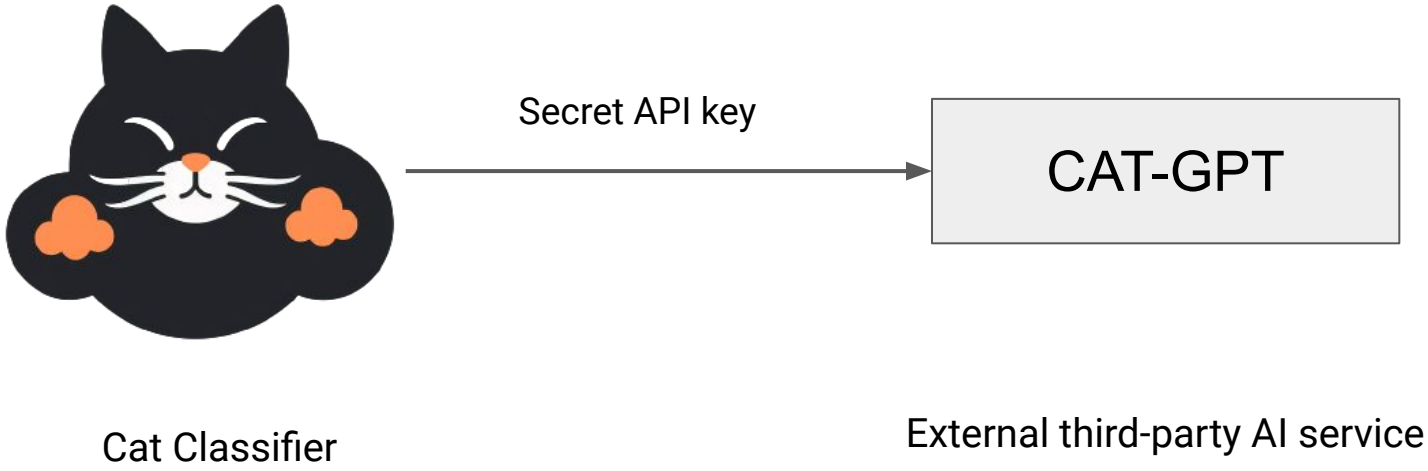
- GKE: Enabling GKE Workload Identity **is enough**
  - prevents pods from accessing node credentials
  - instead, pods get an access token with the permissions of their bound GCP service account

- AWS: IRSA / Azure AD Workload Identity is **not enough** ⚠️
  - Injects a service account on the pod filesystem
  - You still need to explicitly **block IMDS access**

- Exception (by design): pods with host network access

https://learn.microsoft.com/en-us/azure/aks/operator-best-practices-cluster-security?tabs=azure-cli#restrict-access-to-instance-metadata-api
https://aws.github.io/aws-eks-best-practices/security/docs/iam/#restrict-access-to-the-instance-profile-assigned-to-the-worker-node

Moving on! Adding secrets to our application



Bringing cloud secrets into the cluster

external-secrets / **external-secrets** Public

External Secrets Operator reads information from a third-party service like AWS Secrets Manager and automatically injects the values as Kubernetes Secrets.

external-secrets.io/main

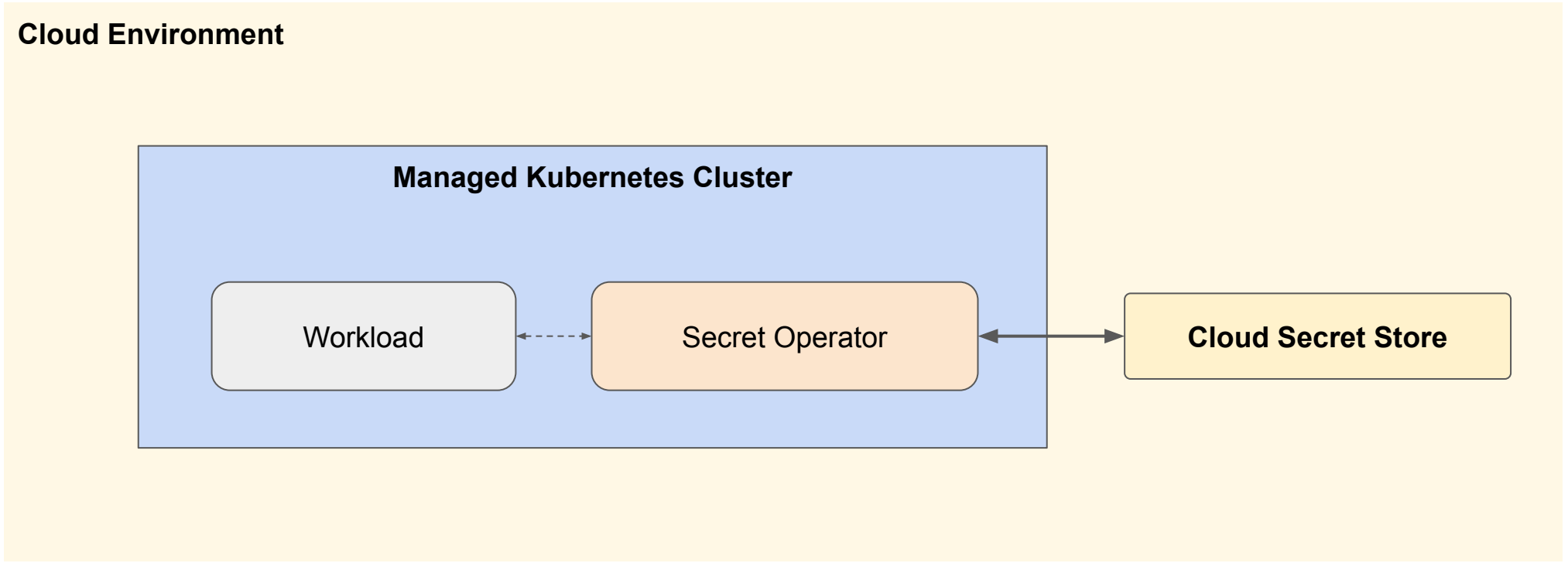
Apache-2.0 license

2.5k stars 461 forks

Kubernetes Secrets Store CSI Driver

Secrets Store CSI Driver for Kubernetes secrets - Integrates secrets stores with Kubernetes via a Container Storage Interface (CSI) volume.

Operator pattern

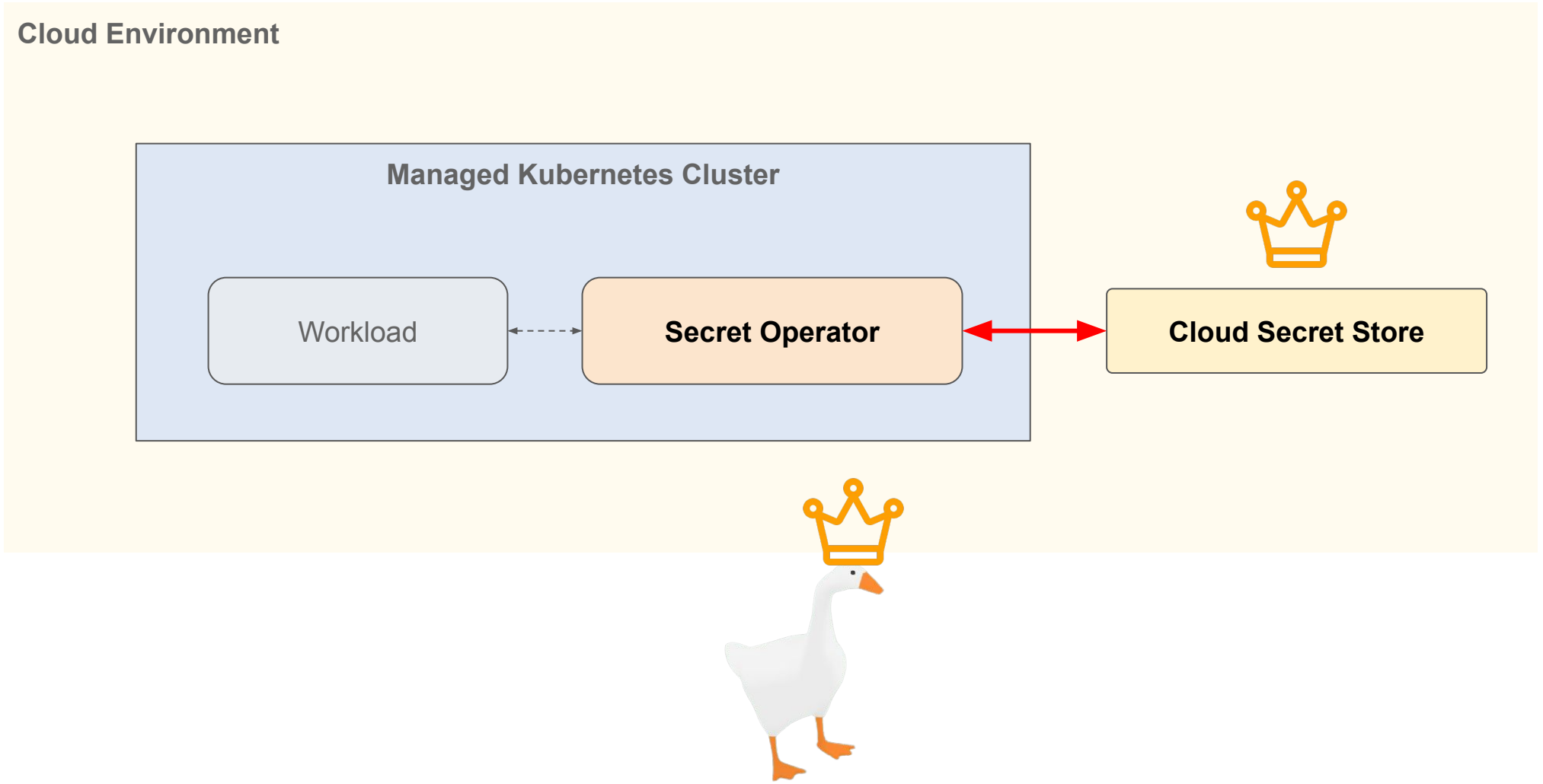


Granting cloud access to the secret operator pods

We first starting by creating a policy named `secrets-reader`:

```
POLICY_ARN=$(aws iam create-policy --policy-name secrets-reader --polic
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:ListSecrets",
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}' | jq -r .Policy.Arn)
```

Granting cloud access to the secret operator pods



Demo: Compromising an Operator and pivoting to the cloud

```
aws sts get-caller-identity
{
    "UserId": "AROA2385J04N5GAMWG44G:i-0464158ab0fe5499",
    "Account": "673001038093",
    "Arn": "arn:aws:sts::673001038093:assumed-role/eksctl-wiz-cluster-nodegroup-ng-NodeInstanceRole-AX6UFF556922/i-046
4158ab0fe5499"
}
kubectl-whoami
system:node:ip-192-168-60-118.eu-west-1.compute.internal
kubectl get pods -A
NAMESPACE          NAME                                                READY   STATUS    RESTARTS   AGE
default            vulnerable-application                              1/1     Running   0          4d10h
external-secrets   external-secrets-66cfb84c9b-kldt9                   1/1     Running   0          10h
external-secrets   external-secrets-cert-controller-7fc8d8d69-s2cq9    1/1     Running   0          10h
external-secrets   external-secrets-webhook-6495c8448b-shslx           1/1     Running   0          10h
kafka              kafka-proxy                                         1/1     Running   0          7d
kafka              kafka-producer                                      1/1     Running   0          7d
kube-system        aws-node-88lnq                                      1/1     Running   0          7d11h
kube-system        coredns-7cc879f8db-gsl6x                            1/1     Running   0          7d11h
kube-system        coredns-7cc879f8db-jxk8s                            1/1     Running   0          7d11h
kube-system        kube-proxy-t7fmx                                    1/1     Running   0          7d11h
microservices      inventory-service                                   1/1     Running   0          7d
microservices      inventory-service-2                                 1/1     Running   0          7d
microservices      rate-limiter-1                                      1/1     Running   0          7d
microservices      rate-limiter-2                                      1/1     Running   0          7d
microservices      rate-limiter-3                                      1/1     Running   0          7d
kubectl create token -n external-secrets external-secrets-sa --bound-object-kind=Pod --bound-object-name extern
al-secrets-66cfb84c9b-kldt9 --bound-object-uid=f8de0e08-06fd-45fe-a75a-1f7fbd903130 --audience sts.amazonaws.com > toke
n
```

Demo summary

1. (Compromised node AWS IAM credentials through the instance metadata service)
2. Enumerated running pods, found External Secrets operator running
3. Created a service account token for the External Secrets pod
4. Exchanged it for AWS credentials for the ExternalSecrets role
5. 

Operators are "bridging the gap" between cluster and cloud

Some are very powerful!

- Manage cloud IAM from the cluster
- Manage infrastructure resources from the cluster

Create IAM role with trust relationship:

```
$ eksctl create iamserviceaccount \
    --cluster "${CLUSTER_NAME}" \
    --region "${AWS_REGION}" \
    --name="${SERVICE_ACCOUNT_NAME}" \
    --namespace="${SERVICE_ACCOUNT_NAMESPACE}" \
    --role-name="${IAM_ROLE_NAME}" \
    --role-only \
    --attach-policy-arn="arn:aws:iam::aws:policy/AdministratorAccess" \
    --approve
```

Minimizing the impact of compromised K8s operators

- Assign minimal cloud privileges to your Kubernetes operators
  - leverage tag-based access control
  - https://github.com/iann0036/iamlive

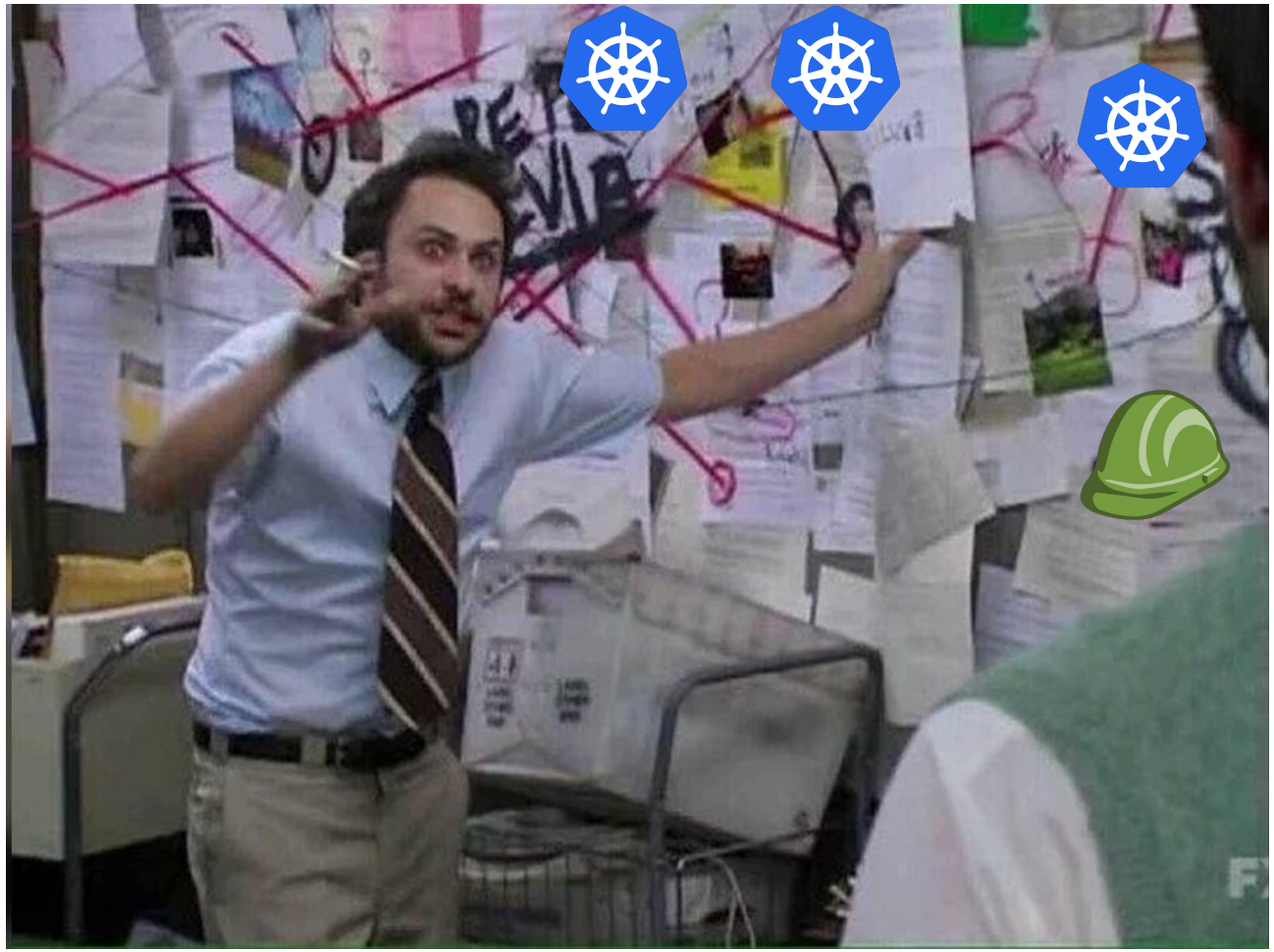
- Know your cluster-to-cloud pivots!

Key Takeaways

Wrapping up - Key takeaways

1. Don't hardcode cloud credentials, use platform-managed authentication
2. Restrict pod access to the instance metadata service
3. Be mindful of cloud permissions you assign to your workloads

Wrapping up - Common challenges



Wrapping up - Common challenges

- What cloud access do my K8s workloads have?
- Do I have hardcoded cloud secrets in my cluster?
- Did I properly restrict pod access to the IMDS?

DataDog / managed-kubernetes-auditing-toolkit

All-in-one auditing toolkit for identifying common security issues in managed Kubernetes environments.

Apache-2.0 license

Managed Kubernetes Auditing Toolkit (MKAT)

Tests passing | go static analysis passing

MKAT is an all-in-one auditing toolkit for identifying common security issues within managed Kubernetes environments. It is focused on AWS EKS at the moment, and will be extended to other managed Kubernetes environments in the future.

Features:

- 🔎 Identify trust relationships between K8s service accounts and AWS IAM roles
- 🔑 Find hardcoded AWS credentials in K8s resources
- 💀 Test if pods can access the AWS Instance Metadata Service (IMDS)

MKAT (Managed Kubernetes Auditing Toolkit)

- Single binary, uses your current K8s authentication context

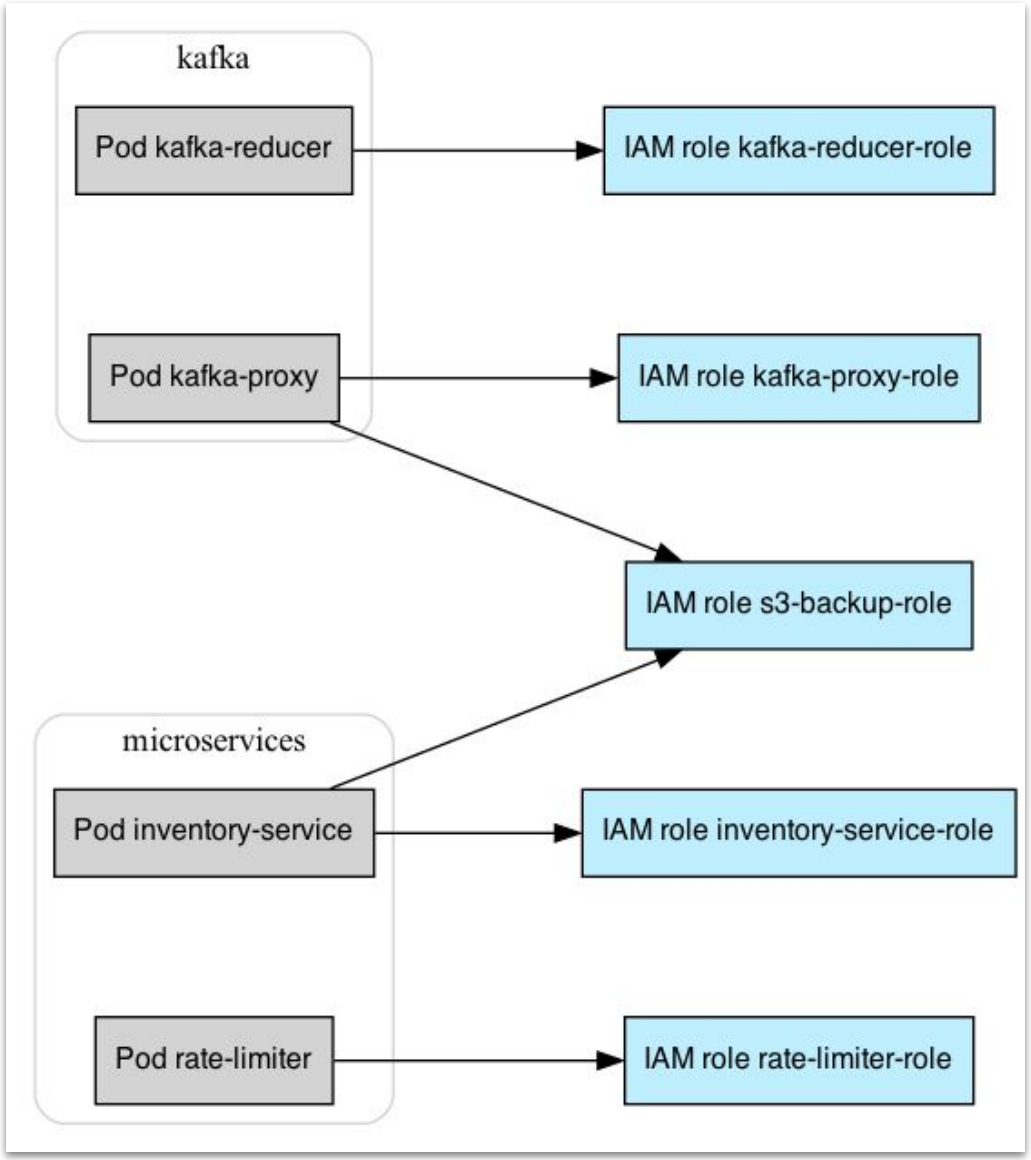
- Features:
  - Identify trust relationships between K8s service accounts and AWS IAM roles
  - Find hardcoded AWS credentials in K8s resources
  - Test if pods can access the AWS Instance Metadata Service (IMDS)

MKAT (Managed Kubernetes Auditing Toolkit)

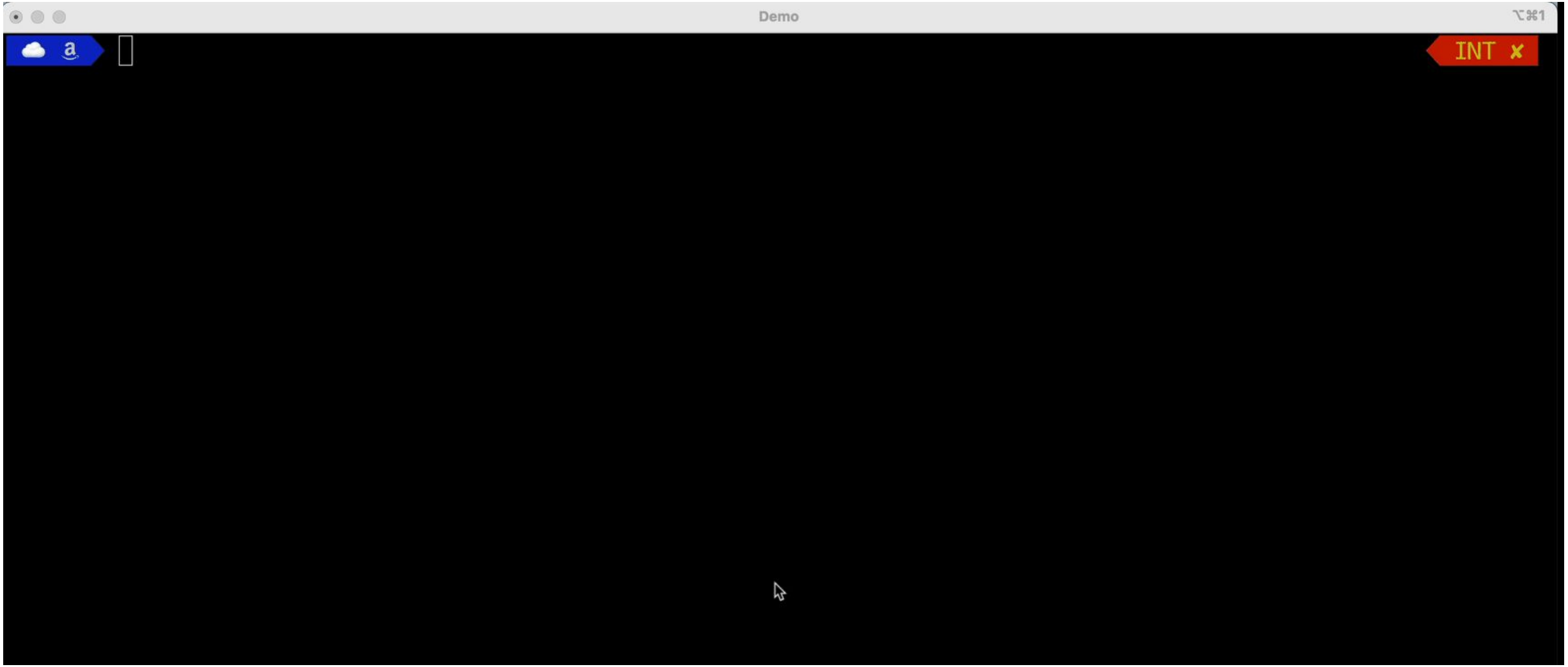
How does MKAT compare to other tools?

Tool	Description
kube-bench	kube-bench is a general-purpose auditing tool for Kubernetes cluster, checking for compliance against the CIS benchmarks
kubiscan	kubiscan focuses on identifying dangerous in-cluster RBAC permissions
peirates	peirates is a generic Kubernetes penetration testing tool. Although it has a `get-aws-token` command that retrieve node credentials from the IMDS, it is not specific to managed K8s environments.
botb	botb is a generic Kubernetes penetration testing tool. It also has a command to retrieve node credentials from the IMDS, but it is not specific to managed K8s environments.
rbac-police	rbac-police focuses on identifying in-cluster RBAC relationships.
kdigger	kdigger is a general-purpose context discovery tool for Kubernetes penetration testing. It does not attempt to be specific to managed K8s environments.
kubeletmein	kubeletmein *is* specific to managed K8s environments. It's an utility to generate a kubeconfig file using the node's IAM credentials, to then use it in a compromised pod.
hardeneks	hardeneks *is* specific to managed K8s environments, but only for EKS. It identifies issues and lack of best practices inside of the cluster, and does not focus on cluster to cloud pivots.

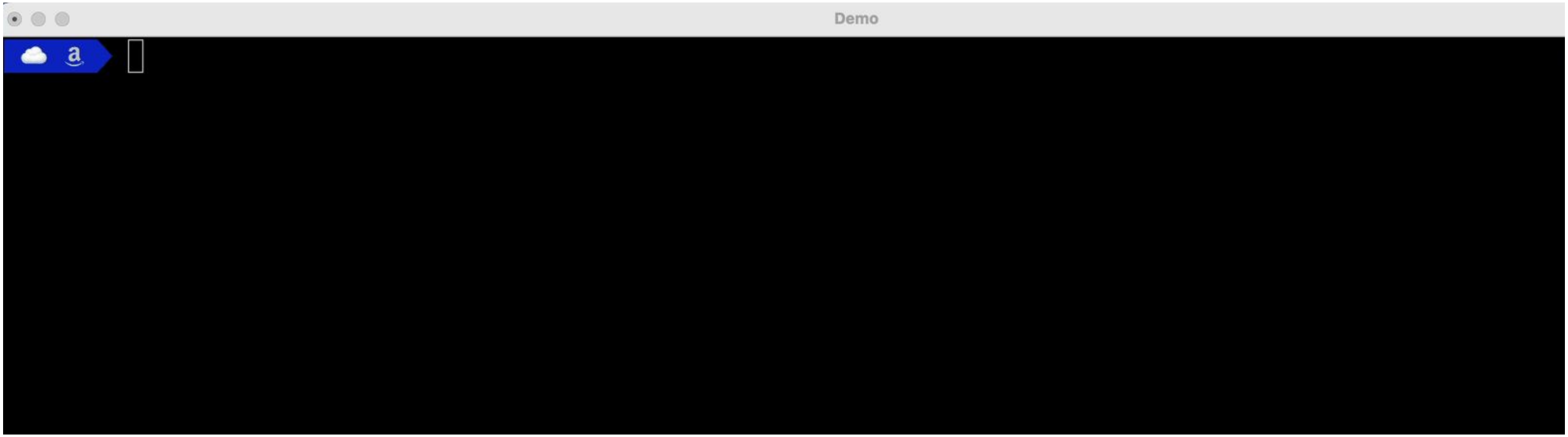
Mapping K8s to AWS role relationships



Find hardcoded AWS secrets



Test IMDS access from pods



Try it out!

```
brew tap datadog/mkat https://github.com/datadog/managed-kubernetes-auditing-toolkit
brew install datadog/mkat/managed-kubernetes-auditing-toolkit
mkat version
```

Roadmap:

- ➢ Identify and visualize pods exposed through an ALB with the ALB Controller
- ➢ Implement support for GKE
- ➢ Add detection of more cloud-specific secrets

KubeCon | CloudNativeCon Europe 2023

Thank you

Slides and feedback!

https://sched.co/1HyZm