

Watermelon Installation Guide

Overview

The Watermelon application can be deployed on any Kubernetes-based platform. Installation artifacts will be shared by the Watermelon team via email, and container images will be made available through a secure private container registry with token-based access at the time of deployment.

Prerequisites

Before beginning the installation, ensure the following components are in place:

AWS EKS

- A functional Amazon Elastic Kubernetes Service cluster

AWS RDS (Postgres)

- PostgreSQL database instance configured and accessible
- Required databases created (Refer to section 1 of Appendix: Database Requirements)

Kubernetes Namespace and Service Account Setup

- **Create the Namespace**
Create a dedicated namespace for the Watermelon application:
`kubectl create namespace watermelon`
- **Create the Service Account**
Apply the service account configuration using the provided YAML file:
 - Navigate to: `<PATH>/PLATFORM/`
 - Deploy using the following commands:

```
kubectl apply -f serviceaccount.yaml
```

This creates a service account named **watermelon** in the **watermelon** namespace. The Watermelon pods will use this service account to access AWS services via Pod Identity.

AWS S3 Buckets

- S3 buckets provisioned (Refer to section 2 of Appendix: S3 Bucket Requirements)

AWS Pod Identity

- Pod Identity properly configured for service account authentication (Refer to section 3 of Appendix: AWS Pod Identity Configuration)

Docker Registry Secret

Create docker registry secret in the Watermelon namespace on the kubernetes cluster

```
kubectrl create secret docker-registry docker-registry-secret
--docker-server=https://index.docker.io/v1/ --docker-username=wtrmlndbs
--docker-password=dckr_pat_pnJlxviT0wMk_1GoVcO-lgrYtrQ
```

Installation Steps for Watermelon:

Unzip the Watermelon-Installer.zip and follow the sequence mentioned below to install the watermelon application.

SEQUENCE 1 (PLATFORM)

Ingress:

- Navigate to: `<PATH>/PLATFORM/ingress.yml`
- Modify the ingress.yml according to your environment. Read the comments section carefully in the ingress.yml and make the changes.
- Deploy using the following commands:

```
kubectrl apply -f ingress.yml
```

- Please note before applying the ingress.yml file ensure the application load balancer configuration on the EKS is already done. In case not done, refer to section 4 of APPENDIX (Application Load Balancer Configuration)

SEQUENCE 2 (PLATFORM)

Vault:

- Deploy the vault application
- Navigate to: `<PATH>/PLATFORM/vault`
- Deploy using the following commands:

```
kubectrl apply -f configmap.yaml
```

```
kubectrl apply -f rbac.yaml
```

```
kubectrl apply -f services.yaml
```

```
kubectrl apply -f vault.yaml
```

If vault is installed successfully, the output should look something like the following:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
vault-0	1/1	Running	0	101d	10.0.23.241	ip-10-0-29-187.ap-south-1.compute.internal	<none>		<none>	

Once the vault is deployed successfully you need to get the root TOKEN. Root token can be fetched from vault-0 logs. Command to fetch the logs.

```
kubectrl logs -f vault-0
```

Inside the logs look for a string something like this `"hvs.7fn0Tkj9NmjEttZxl6uvPgKs"`. Keep this token handy, as this needs to be keyed into the `wmconfig.yaml` file.

```

2025-12-31T02:12:04.416Z [INFO] core.cluster-listener.tcp: starting listener: listener_address=[:]:8201
2025-12-31T02:12:04.416Z [INFO] core.cluster-listener: serving cluster requests: cluster_listen_address=[:]:8201
2025-12-31T02:12:04.417Z [INFO] core: post-unseal setup starting
2025-12-31T02:12:04.423Z [INFO] core: loaded wrapping token key
2025-12-31T02:12:04.423Z [INFO] core: successfully setup plugin runtime catalog
2025-12-31T02:12:04.424Z [INFO] core: successfully setup plugin catalog: plugin-directory=""
2025-12-31T02:12:04.435Z [INFO] core: successfully mounted: type=system version="v1.21.1+builtin.vault" path=sys/ namespace="ID: root. Path: "
2025-12-31T02:12:04.440Z [INFO] core: successfully mounted: type=identity version="v1.21.1+builtin.vault" path=identity/ namespace="ID: root. Path: "
2025-12-31T02:12:04.443Z [INFO] core: successfully mounted: type=kv version="v0.25.0+builtin" path=secret/ namespace="ID: root. Path: "
2025-12-31T02:12:04.443Z [INFO] core: successfully mounted: type=cubbyhole version="v1.21.1+builtin.vault" path=cubbyhole/ namespace="ID: root. Path: "
2025-12-31T02:12:04.448Z [INFO] core: successfully mounted: type=token version="v1.21.1+builtin.vault" path=token/ namespace="ID: root. Path: "
2025-12-31T02:12:04.450Z [INFO] rollback: Starting the rollback manager with 256 workers
2025-12-31T02:12:04.450Z [INFO] rollback: starting rollback manager
2025-12-31T02:12:04.450Z [INFO] core: restoring leases
2025-12-31T02:12:04.451Z [INFO] expiration: lease restore complete
2025-12-31T02:12:04.453Z [INFO] identity: entities restored
2025-12-31T02:12:04.453Z [INFO] identity: groups restored
2025-12-31T02:12:04.458Z [INFO] core: usage gauge collection is disabled
2025-12-31T02:12:04.464Z [INFO] core: post-unseal setup complete
2025-12-31T02:12:04.464Z [INFO] core: vault is unsealed
hvs.7Fn0Tkj9NmjEttZx16uvPgKs

```

SEQUENCE 3 (PLATFORM)

Configmap update

- Navigate to: `<PATH>/PLATFORM/`
- Update the `wmconfig.yaml` with all parameters mentioned in the comments section of the `wmconfig.yaml`

```
kubectl apply -f wmconfig.yaml
```

SEQUENCE 4 (PLATFORM)

Opensearch

- Navigate to: `<PATH>/PLATFORM/opensearch`
- Deploy using the following commands:

```
kubectl apply -f configmap.yaml
```

```
kubectl apply -f poddisruptionbudget.yaml
```

```
kubectl apply -f service.yaml
```

```
kubectl apply -f statefulset.yaml
```

If opensearch is installed successfully, the output should look something like the following:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
opensearch-cluster-master-0	1/1	Running	0	185d	10.0.10.140	ip-10-0-3-23.ap-south-1.compute.internal	<none>	<none>
opensearch-cluster-master-1	1/1	Running	0	101d	10.0.28.49	ip-10-0-29-187.ap-south-1.compute.internal	<none>	<none>
opensearch-cluster-master-2	1/1	Running	0	178d	10.0.42.10	ip-10-0-38-69.ap-south-1.compute.internal	<none>	<none>

SEQUENCE 5 (PLATFORM)

Rabbitmq

- Navigate to: `<PATH>/PLATFORM/rabbitmq`
- Deploy using the following commands:

```
kubectl apply -f serviceaccount.yaml
```

```
kubectl apply -f role.yaml
```

```
kubectl apply -f rolebinding.yaml
```

```
kubectl apply -f configmap.yaml
```

```
kubectl apply -f secret.yaml
```

```
kubectl apply -f service-headless.yaml
```

```
kubectl apply -f service.yaml
```

```
kubectl apply -f statefulset.yaml
```

If rabbitmq is installed successfully, the output should look something like the following:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
rabbitmq-0	1/1	Running	0	185d	10.0.13.50	ip-10-0-3-23.ap-south-1.compute.internal	<none>	<none>

SEQUENCE 6 (PLATFORM)

Selenium Grid:

- Navigate to the directory: `<PATH>/PLATFORM/selenium-grid`
- Locate the values.yml file and make the necessary changes as per the comments in the file.
- Deploy using the command:
`helm install selenium .`

If selenium-grid is installed successfully, the output should look something like the following:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
keda-operator-5b74946745-nwk25	1/1	Running	0	94d	10.0.8.100	ip-10-0-3-23.ap-south-1.compute.internal	<none>	<none>
keda-operator-metrics-apiserver-c95887576-s89bq	1/1	Running	0	94d	10.0.30.235	ip-10-0-29-187.ap-south-1.compute.internal	<none>	<none>
selenium-hub-9bc5746d8-vnb7v	1/1	Running	0	94d	10.0.33.203	ip-10-0-43-146.ap-south-1.compute.internal	<none>	<none>
selenium-node-chrome-865ccb699-9mk8w	2/2	Running	0	94d	10.0.20.101	ip-10-0-29-187.ap-south-1.compute.internal	<none>	<none>
selenium-node-edge-777f99bcfb-cdxzv	2/2	Running	0	94d	10.0.35.169	ip-10-0-38-69.ap-south-1.compute.internal	<none>	<none>

SEQUENCE 7 (CORE)

wmconsul:

- Navigate to: `<PATH>/CORE/wmconsul`
- Deploy using the following commands:

`kubectl apply -f wmconsul.yml`

If wmconsul is installed successfully, the output should look something like the following:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
wmconsul-0	1/1	Running	0	163d	10.0.42.94	ip-10-0-43-146.ap-south-1.compute.internal	<none>	<none>

SEQUENCE 8 (CORE)

wmkeycloak:

- Navigate to: `<PATH>/CORE/wmkeycloak`
- Deploy using the following commands:
`kubectl apply -f watermelon-realm.yml`
`kubectl apply -f wmkeycloak.yml`

If wmkeycloak is installed successfully, the output should look something like the following:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
wmkeycloak-0	1/1	Running	0	178d	10.0.9.216	ip-10-0-14-24.ap-south-1.compute.internal	<none>	<none>

Before proceeding to next step ensure all the above services are up and running

SEQUENCE 9 (CORE)

wmgateway:

- Navigate to: `<PATH>/CORE/wmgateway`
- Deploy using the command:
`kubectl apply -f wmgateway.yml`

If wmgateway is installed successfully, the output should look something like the following:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
wmgateway-7d5f7545bc-lj2v6	1/1	Running	0	49d	10.0.28.206	ip-10-0-29-187.ap-south-1.compute.internal	<none>	<none>

SEQUENCE 10 (CORE)

wmmeta:

- Navigate to: <PATH>/CORE/wmmeta
- Deploy using the command:
kubectl apply -f wmmeta.yml

If wmmeta is installed successfully, the output should look something like the following:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
wmmeta-6684cc8c4-f9tms	1/1	Running	0	49d	10.0.35.203	ip-10-0-43-146.ap-south-1.compute.internal	<none>	<none>

SEQUENCE 11 (AUTONOMOUS FUNCTIONAL TESTING)

wmuitestcontroller:

- Navigate to: <PATH>/AFT/wmuitestcontroller
- Deploy using the command:
kubectl apply -f wmuitestcontroller.yml

If wmuitestcontroller is installed successfully, the output should look something like the following:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
wmuitestcontroller-6bc6857544-5ns2z	1/1	Running	0	49d	10.0.22.77	ip-10-0-21-72.ap-south-1.compute.internal	<none>	<none>
wmuitestcontroller-6bc6857544-6ns5z	1/1	Running	0	49d	10.0.22.77	ip-10-0-21-72.ap-south-1.compute.internal	<none>	<none>

wmtestdatafactory:

- Navigate to: <PATH>/AFT/wmtestdatafactory
- Deploy using the command:
kubectl apply -f wmtestdatafactory.yml

If wmtestdatafactory is installed successfully, the output should look something like the following:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
wmtestdatafactory-74589868f4-h7vtj	1/1	Running	0	49d	10.0.9.33	ip-10-0-3-23.ap-south-1.compute.internal	<none>	<none>
wmtestdatafactory-74589868f4-18wuk	1/1	Running	0	49d	10.0.9.33	ip-10-0-3-23.ap-south-1.compute.internal	<none>	<none>

wmaiengine:

- Navigate to: <PATH>/AFT/wmaiengine
- Deploy using the command:
kubectl apply -f wmaiengine.yml

If wmaiengine is installed successfully, the output should look something like the following:

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
wmaiengine-56f65667ff-mpg4p	1/1	Running	0	126d	10.0.5.39	ip-10-0-14-24.ap-south-1.compute.internal	<none>	<none>

wmuitestexecutor:

- Navigate to: `<PATH>/AFT/wmuitestexecutor`
- Deploy using the command:
`kubectl apply -f wmuitestexecutor-mobile.yml`
`kubectl apply -f wmuitestexecutor-web.yml`

If `wmuitestexecutor` is installed successfully, the output should look something like the following:

```

NAME                READY   STATUS    RESTARTS   AGE   IP              NODE                                                                 NOMINATED NODE   READINESS GATES
wmuitestexecutor-web-0  1/1    Running  0           33d   10.0.26.71     ip-10-0-21-72.ap-south-1.compute.internal   <none>           <none>
wmuitestexecutor-web-1  1/1    Running  0           33d   10.0.12.50     ip-10-0-14-24.ap-south-1.compute.internal   <none>           <none>
wmuitestexecutor-web-2  1/1    Running  0           33d   10.0.41.76     ip-10-0-38-69.ap-south-1.compute.internal   <none>           <none>
wmuitestexecutor-mob-0  1/1    Running  0           33d   10.0.42.36     ip-10-0-43-146.ap-south-1.compute.internal  <none>           <none>

```

SEQUENCE 12

wmebonboarding: (SLO MANAGEMENT)

- Navigate to: `<PATH>/SLO/wmebonboarding`
- Deploy using the command:
`kubectl apply -f wmebonboarding.yml`

If `wmebonboarding` is installed successfully, the output should look something like the following:

```

[ec2-user@ip-172-31-47-218 ~]$ kubectl get pods wmebonboarding-0 -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP              NODE                                                                 NOMINATED NODE   READINESS GATES
wmebonboarding-0  1/1    Running  0           13d   172.31.33.202   ip-172-31-42-117.ap-south-1.compute.internal   <none>           <none>
[ec2-user@ip-172-31-47-218 ~]$

```

wmebexecuter:

- Navigate to: `<PATH>/SLO/wmebexecuter`
- Deploy using the command:
`kubectl apply -f wmebexecuter.yml`

If `wmebexecuter` is installed successfully, the output should look something like the following:

```

[ec2-user@ip-172-31-47-218 ~]$ kubectl get pods wmebexecuter-0 -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP              NODE                                                                 NOMINATED NODE   READINESS GATES
wmebexecuter-0     1/1    Running  0           6d9h  172.31.39.219   ip-172-31-39-150.ap-south-1.compute.internal   <none>           <none>
[ec2-user@ip-172-31-47-218 ~]$

```

wmebstatsservice:

- Navigate to: `<PATH>/SLO/wmebstatsservice`
- Deploy using the command:
`kubectl apply -f wmebstatsservice.yml`

If `wmebstatsservice` is installed successfully, the output should look something like the following:

```

[ec2-user@ip-172-31-47-218 ~]$ kubectl get pods wmebstatsservice-0 -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP              NODE                                                                 NOMINATED NODE   READINESS GATES
wmebstatsservice-0  1/1    Running  0           8d    172.31.8.55     ip-172-31-5-245.ap-south-1.compute.internal   <none>           <none>
[ec2-user@ip-172-31-47-218 ~]$

```

wmebnotifierservice:

- Navigate to: `<PATH>/SLO/wmebnotifierservice`
- Deploy using the command:
`kubectl apply -f wmebnotifierservice.yml`

If `wmebnotifierservice` is installed successfully, the output should look something like the following:

```

[ec2-user@ip-172-31-47-218 ~]$ kubectl get pods wmebnotifierservice-0 -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP              NODE                                                                 NOMINATED NODE   READINESS GATES
wmebnotifierservice-0  1/1    Running  0           15d   172.31.35.159   ip-172-31-39-150.ap-south-1.compute.internal   <none>           <none>
[ec2-user@ip-172-31-47-218 ~]$

```

SEQUENCE 13 (LIGHTNING API)

wmtestopsctrl:

- Navigate to: `<PATH>/LAPI/wmtestopsctrl`
- Deploy using the command:
`kubectly apply -f wmtestopsctrl.yml`

If `wmtestopsctrl` is installed successfully, the output should look something like the following:

```
[ec2-user@ip-172-31-47-218 ~]$ kubectl get pods wmtestopsctrl-0 -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP              NODE                                                                 NOMINATED NODE   READINESS GATES
wmtestopsctrl-0 1/1     Running   0           4d1h  172.31.2.5      ip-172-31-8-243.ap-south-1.compute.internal   <none>           <none>
```

wmtestopsexecutor:

- Navigate to: `<PATH>/LAPI/wmtestopsexecutor`
- Deploy using the command:
`kubectly apply -f wmtestopsexecutor.yml`

If `wmtestopsexecutor` is installed successfully, the output should look something like the following:

```
[ec2-user@ip-172-31-47-218 ~]$ kubectl get pods wmtestopsexecutor-0 -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP              NODE                                                                 NOMINATED NODE   READINESS GATES
wmtestopsexecutor-0 1/1     Running   0           4d1h  172.31.6.241    ip-172-31-5-245.ap-south-1.compute.internal   <none>           <none>
```

SEQUENCE 14 (CHAOS ENGINEERING)

wmchaosdiscovery:

- Navigate to: `<PATH>/CHAOS/wmchaosdiscovery`
- Deploy using the command:
`kubectly apply -f wmchaosdiscovery.yml`

If `wmchaosdiscovery` is installed successfully, the output should look something like the following:

```
[ec2-user@ip-172-31-47-218 ~]$ kubectl get pods wmchaosdiscovery-595977ff78-2vrk7 -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP              NODE                                                                 NOMINATED NODE   READINESS GATES
wmchaosdiscovery-595977ff78-2vrk7 1/1     Running   0           27d   172.31.38.68    ip-172-31-39-150.ap-south-1.compute.internal   <none>           <none>
```

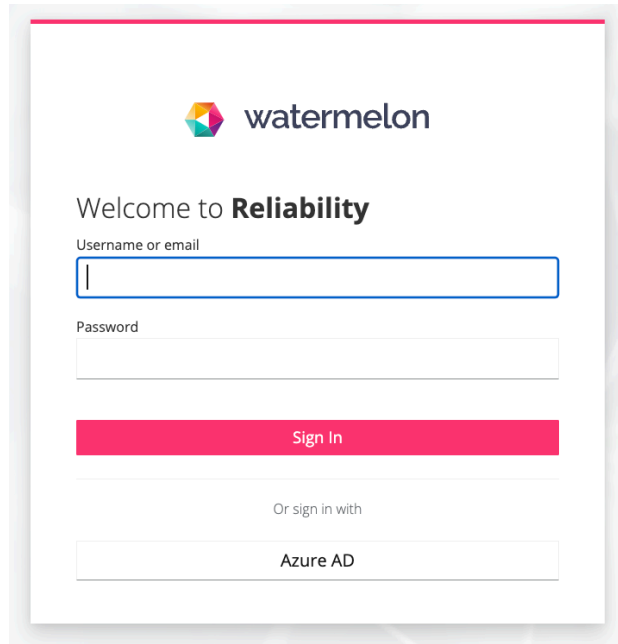
wmchaoscontroller:

- Navigate to: `<PATH>/CHAOS/wmchaoscontroller`
- Deploy using the command:
`kubectly apply -f wmchaoscontroller.yml`

If `wmchaoscontroller` is installed successfully, the output should look something like the following:

```
[ec2-user@ip-172-31-47-218 ~]$ kubectl get pods wmchaoscontroller-0 -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP              NODE                                                                 NOMINATED NODE   READINESS GATES
wmchaoscontroller-0 1/1     Running   0           89d   172.31.32.101   ip-172-31-42-117.ap-south-1.compute.internal   <none>           <none>
```

On successful completion and validation of all the above steps, you may use the Watermelon Gateway URL from the ALB to access the platform. If you see the below screen on your browser, that indicates that the deployment is completed successfully. If you encounter any issues, please reach out to your Watermelon representative for support.



APPENDIX

Section 1

Database Requirements

The following databases must be created in your AWS RDS PostgreSQL instance

Database Names
wmkeycloak
wmgateway
wmmeta
wmuitestcontroller
wmtestdatafactory
wmtestopsctrl
wmtestopsexecutor
wmchaos
wmebservices

Section 2

S3 Bucket Requirements

The following S3 buckets must be provisioned:

Bucket Name
wm-test-scm-repository
wm-ui-test-page-source
wm-ft-tmp
wm-ui-test-scripts
wm-ui-test-apk-ipa
wm-meta
wm-ui-test-resource
wm-ui-test-videos
wm-ui-test-har
wm-ui-test-screenshots
wm-ui-test-browser-logs
wm-ai-chat-files
wm-ai-doc-test-gen

Section 3

AWS Pod Identity Configuration

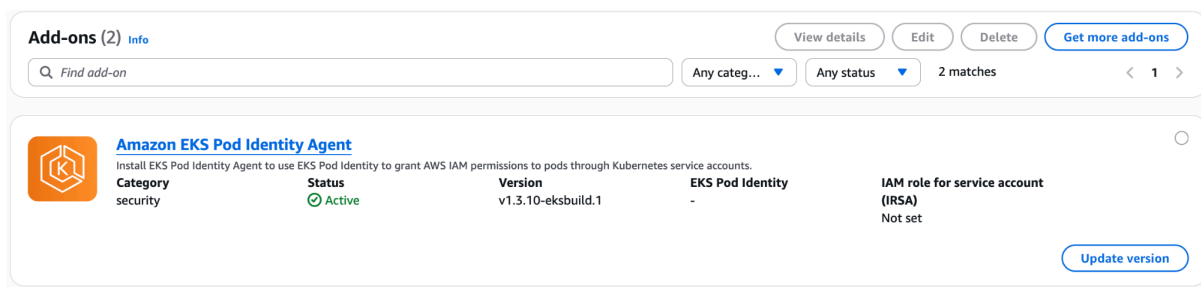
AWS Pod Identity enables your Kubernetes pods to authenticate with AWS services using IAM roles. Follow these steps to configure it:

Configuration Steps

1. Enable Required EKS Add-ons

Ensure the following add-ons are installed and enabled in your EKS cluster:

- **EKS Pod Identity Agent:** This add-on is required for pods to assume IAM roles using EKS Pod Identity



The screenshot shows the AWS EKS Add-ons console. At the top, it says "Add-ons (2) Info" with buttons for "View details", "Edit", "Delete", and "Get more add-ons". Below this is a search bar with "Find add-on" and filters for "Any categ...", "Any status", and "2 matches". The main content area shows the "Amazon EKS Pod Identity Agent" add-on. It includes an icon, the name, a description: "Install EKS Pod Identity Agent to use EKS Pod Identity to grant AWS IAM permissions to pods through Kubernetes service accounts.", and a table of details:

Category	Status	Version	EKS Pod Identity	IAM role for service account (IRSA)
security	Active	v1.3.10-eksbuild.1	-	Not set

There is also an "Update version" button at the bottom right of the add-on card.

2. Create IAM Policy for S3 Access

Create an IAM policy that grants the necessary S3 permissions for the Watermelon application.

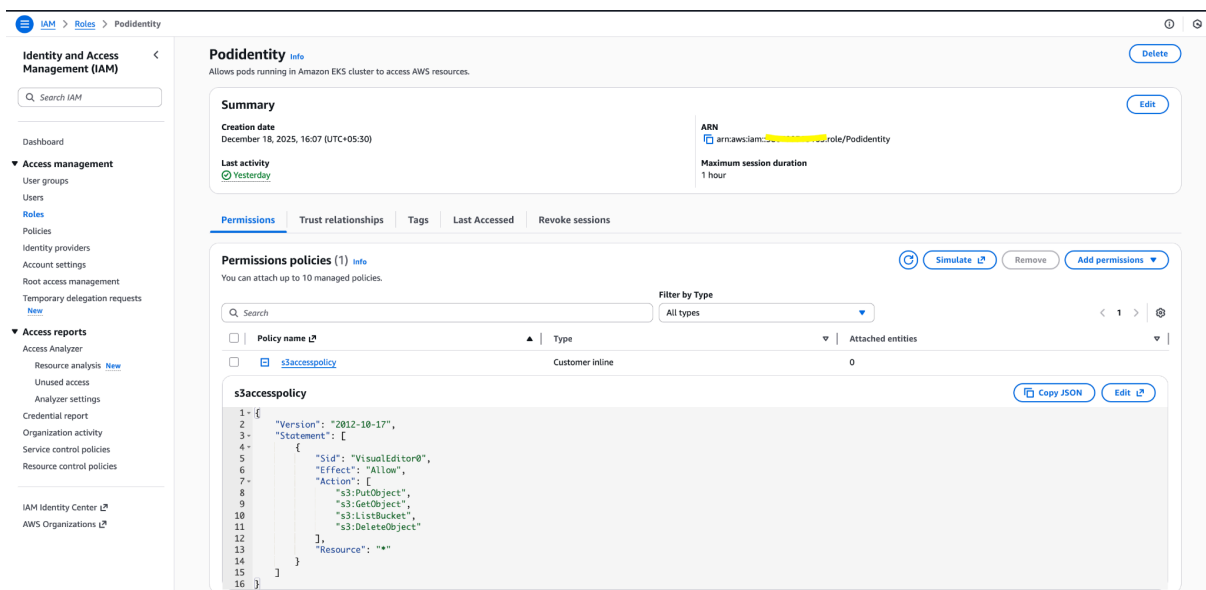
Policy Configuration

```

json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::your-bucket-name/*",
        "arn:aws:s3:::your-bucket-name"
      ]
    }
  ]
}

```

Replace **your-bucket-name** with the actual S3 bucket name(s) that Watermelon needs to access.



The screenshot shows the AWS IAM console interface for a PodIdentity role. The left sidebar contains navigation options for IAM and Access Management. The main content area displays the role's summary, including its creation date and ARN. Below the summary, the 'Permissions' tab is active, showing a list of permissions policies. One policy, 's3accesspolicy', is attached to the role. The policy's JSON configuration is displayed in a code editor, showing the same permissions as the JSON block above: 's3:GetObject', 's3:PutObject', 's3:DeleteObject', and 's3:ListBucket' on the resource 'arn:aws:s3:::*/*' and 'arn:aws:s3:::*'.

Attach Policy to IAM Role

After creating the policy, attach it to the IAM role that will be used by the Pod Identity association.

3. Create Pod Identity Association

Create a Pod Identity association that links your Kubernetes service account to the IAM role with S3 permissions.

Configuration Details:

- **Namespace:** Specify the Kubernetes namespace (watermelon) where Watermelon is deployed.
- **Service Account:** Use the service account (watermelon) for the Watermelon application.
- **IAM Role:** Associate the IAM role that has the S3 access policy attached

The screenshot shows the Amazon Elastic Kubernetes Service console for a cluster named 'presale-demo-eks'. The 'Access' tab is selected, displaying the 'IAM access entries' section. There are 5 IAM access entries listed. The first entry is for the 'aws-service-role/eks.amazonaws.com/AWSServiceRoleForAmazonEKS' role, which is associated with the 'system:node' group. The second entry is for the 'presale-demo-eks-role' role, which is associated with the 'system:node' group. The third, fourth, and fifth entries are for 'presale-demo-jumphostrole' roles, which are associated with the 'AmazonEKSClusterAdminPolicy'.

IAM principal ARN	Type	Username	Group names	Access policies
arn:aws:iam::000000000000:role/eks-service-role/eks.amazonaws.com/AWSServiceRoleForAmazonEKS	Standard	arn:aws:sts::000000000000:assumed-role/AWSServiceRoleForAmazonEKS/[(SessionName)]		AmazonEKSClusterInsightsPolicy, AmazonEKSEventPolicy
arn:aws:iam::000000000000:role/presale-demo-eks-role	EC2 Linux	system:node:[EC2PrivateDNSName]	system:nodes	
arn:aws:iam::000000000000:role/presale-demo-jumphostrole	Standard	arn:aws:sts::000000000000:assumed-role/presale-demo-jumphostrole/[(SessionName)]		AmazonEKSClusterAdminPolicy
arn:aws:iam::000000000000:role/presale-demo-jumphostrole	Standard	arn:aws:iam::000000000000:role/presale-demo-jumphostrole		AmazonEKSClusterAdminPolicy
arn:aws:iam::000000000000:role/presale-demo-jumphostrole	Standard	arn:aws:iam::000000000000:role/presale-demo-jumphostrole		AmazonEKSClusterAdminPolicy

Below the IAM access entries, there is a section for 'Pod Identity associations' with 1 association. The association is for the 'a-b4kk73pwmkcre2d' role, which is associated with the 'arn:aws:iam::000000000000:role/PodIdentity' role. The target IAM role is '-', the namespace is 'poc-env', and the service account is 's3access'.

3. Add EKSAUTH policy to nodegroup IAM policy

To enable the node group to support Pod Identity, you need to add a custom authentication policy to the node group's IAM role.

Steps:

1. Navigate to the IAM console
2. Locate the IAM role associated with your EKS node group
3. Create below custom policy and attach to nodegroup IAM policy

The screenshot shows the AWS IAM console 'Permissions policies' page. The 'eksauth' policy is selected. The policy is a customer inline policy. The policy definition is as follows:

```

1- {
2-   "Version": "2012-10-17",
3-   "Statement": [
4-     {
5-       "Sid": "VisualEditor0",
6-       "Effect": "Allow",
7-       "Action": "eks-auth:*",
8-       "Resource": "*"
9-     }
10-  ]
11- }

```

Step 1
 Modify permissions in eksauth
 Step 2
 Review and save

Modify permissions in eksauth [Info](#)

Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.

Policy editor Visual JSON Actions

EKS Auth All actions

Specify what actions can be performed on specific resources in EKS Auth.

Actions allowed

Specify actions from the service to be allowed.

Filter Actions

Manual actions | [Add actions](#)

All EKS Auth actions (eks-auth*)

Access level

Read (Selected 1/1)

All read actions

AssumeRoleForPodIdentity [Info](#)

Resources

Specified resource ARNs for these actions.

All resources

Request conditions - optional

Actions on resources are allowed or denied only when these conditions are met.

[+ Add more permissions](#)

Security: 0 Errors: 0 Warnings: 0 Suggestions: 0

Cancel **Next**

Permissions Trust relationships Tags (6) Last Accessed Revoke sessions

Permissions policies (3) [Info](#)

You can attach up to 10 managed policies.

Search Filter by Type: All types

Policy name i	Type	Attached entities
<input type="checkbox"/> AmazonSSMManagedInstanceCore	AWS managed	8
<input type="checkbox"/> eksauth	Customer inline	0
<input type="checkbox"/> presalesdemo-eks-restricted-policy	Customer inline	0

▶ **Permissions boundary** (not set)

▼ **Generate policy based on CloudTrail events**

You can generate a new policy based on the access activity for this role, then customize, create, and attach it to this role. AWS uses your CloudTrail events to identify the services and actions used and generate a policy. [Learn more](#)

[Generate policy](#)

No requests to generate a policy in the past 7 days

ADDITIONAL CONFIGURATIONS

On "wm-ui-test-resource" bucket add the following CORS permission

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET",
      "HEAD"
    ],
    "AllowedOrigins": [
      "https://wmsandbox5.watermelon.us" → Replace the URL with your actual Watermelon
      application endpoint..
    ],
    "ExposeHeaders": []
  }
]
```

Section 4

Application Load Balancer Configuration

```
[ec2-user@ip-172-31-47-83 ALB]$ curl -O
https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/install
/iam_policy.json
```

```
% Total % Received % Xferd Average Speed Time Time Time Current
      Dload Upload Total Spent Left Speed
100 8386 100 8386 0 0 72656 0 --:--:-- --:--:-- --:--:-- 72921
```

```
[ec2-user@ip-172-31-47-83 ALB]$ aws iam create-policy --policy-name
AWSLoadBalancerControllerIAMPolicy --policy-document file://iam_policy.json
```

```
{
  "Policy": {
    "PolicyName": "AWSLoadBalancerControllerIAMPolicy",
    "PolicyId": "ANPA5FTZFKY6334ID7A6C",
    "Arn": "arn:aws:iam::905XXXXXX:policy/AWSLoadBalancerControllerIAMPolicy",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2024-03-24T10:39:00+00:00",
    "UpdateDate": "2024-03-24T10:39:00+00:00"
  }
}
```

```
[ec2-user@ip-172-31-47-83 ALB]$ eksctl utils associate-iam-oidc-provider --region=us-east-2
--cluster=wm-sandbox-poc-eks --approve
```

```
2024-03-24 10:41:40 [i] will create IAM Open ID Connect provider for cluster "wm-sandbox-poc-eks" in
"us-east-2"
2024-03-24 10:41:40 [✓] created IAM Open ID Connect provider for cluster "wm-sandbox-poc-eks" in
"us-east-2"
```

```
[ec2-user@ip-172-31-47-83 ALB]$ eksctl create iamserviceaccount --cluster=wm-sandbox-poc-eks
--namespace=kube-system --name=aws-load-balancer-controller --role-name
AmazonEKSLoadBalancerControllerRole
--attach-policy-arn=arn:aws:iam::905XXXXXX:policy/AWSLoadBalancerControllerIAMPolicy --approve
--region=us-east-2
```

```
2024-03-24 10:42:51 [i] 1 iamserviceaccount (kube-system/aws-load-balancer-controller) was included (based
on the include/exclude rules)
```

```
2024-03-24 10:42:51 [!] serviceaccounts that exist in Kubernetes will be excluded, use
--override-existing-serviceaccounts to override
```

```
2024-03-24 10:42:51 [i] 1 task: {
```

```
  2 sequential sub-tasks: {
```

```
    create IAM role for serviceaccount "kube-system/aws-load-balancer-controller",
    create serviceaccount "kube-system/aws-load-balancer-controller",
```

```

} }2024-03-24 10:42:51 [i] building iamserviceaccount stack
"eksctl-wm-sandbox-poc-eks-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2024-03-24 10:42:51 [i] deploying stack
"eksctl-wm-sandbox-poc-eks-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2024-03-24 10:42:51 [i] waiting for CloudFormation stack
"eksctl-wm-sandbox-poc-eks-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2024-03-24 10:43:21 [i] waiting for CloudFormation stack
"eksctl-wm-sandbox-poc-eks-addon-iamserviceaccount-kube-system-aws-load-balancer-controller"
2024-03-24 10:43:22 [i] created serviceaccount "kube-system/aws-load-balancer-controller"

```

```

[ec2-user@ip-172-31-47-83 ALB]$ helm repo add eks https://aws.github.io/eks-charts
"eks" has been added to your repositories

```

```

[ec2-user@ip-172-31-47-83 ALB]$ helm repo update eks
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "eks" chart repository
Update Complete. ✨Happy Helming!✨

```

```

[ec2-user@ip-172-31-47-83 ALB]$ helm install aws-load-balancer-controller
eks/aws-load-balancer-controller -n kube-system --set clusterName=wm-sandbox-poc-eks --set
serviceAccount.create=false --set serviceAccount.name=aws-load-balancer-controller --set
region=us-east-2 --set vpcId=vpc-0c768c78dXXXXXX

```

```

NAME: aws-load-balancer-controller
LAST DEPLOYED: Sun Mar 24 10:46:11 2024
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
AWS Load Balancer controller installed!

```

```

[ec2-user@ip-172-31-47-83 ALB]$ kubectl get pods -n kube-system

```

NAME	READY	STATUS	RESTARTS	AGE
aws-load-balancer-controller-84b57b7c56-lf5hw	1/1	Running	0	227d
aws-load-balancer-controller-84b57b7c56-s8pkv	1/1	Running	0	227d