Installing Automic Automation Kubernetes Edition v21

How to deploy to Azure

Version 1.1

Contents

Chapter 1: Introduction 4

Chapter 2: Create the PostgreSQL Database for Automic 5

Chapter 3: Create the Secrets 6
- 3.1 Automic ImagePullSecret used to retrieve the images from GCR 6
- 3.2 DB secret with the connection information 6
- 3.3 Client 0 secret with pre-set credentials 6

Chapter 4: Create a Public IP Address 7

Chapter 5: Create an Ingress Controller 8

Chapter 6: Configure TLS certificates 9

Chapter 7: Deploy AAKE in the AKS Cluster 11
- 7.1 Download the AAKE zip package and install the Automic Helm Plugin and Helm chart 11
- 7.2 Configure the Ingress in values.yaml 11
- 7.3 Install AAKE using Helm 11

Chapter 8: Expose the Cluster to the outside world 13
- 8.1 Expose the AWI, JCP WS and JCP REST services 13
- 8.2 Expose the CP Services 14

Chapter 9: Connect Agents via TCP or HTTPS Load Balancer 15
- 9.1 Connect TLS-enabled agents 15
- 9.2 Connect non-TLS agents via TLS Gateway 15
- 9.3 Connect non-TLS agents via TCP Load Balancer 16

Chapter 1: Introduction

The Azure Kubernetes Service (AKS) offered by Microsoft on the Azure platform can be used to deploy and manage the Automic Automation Kubernetes Edition.

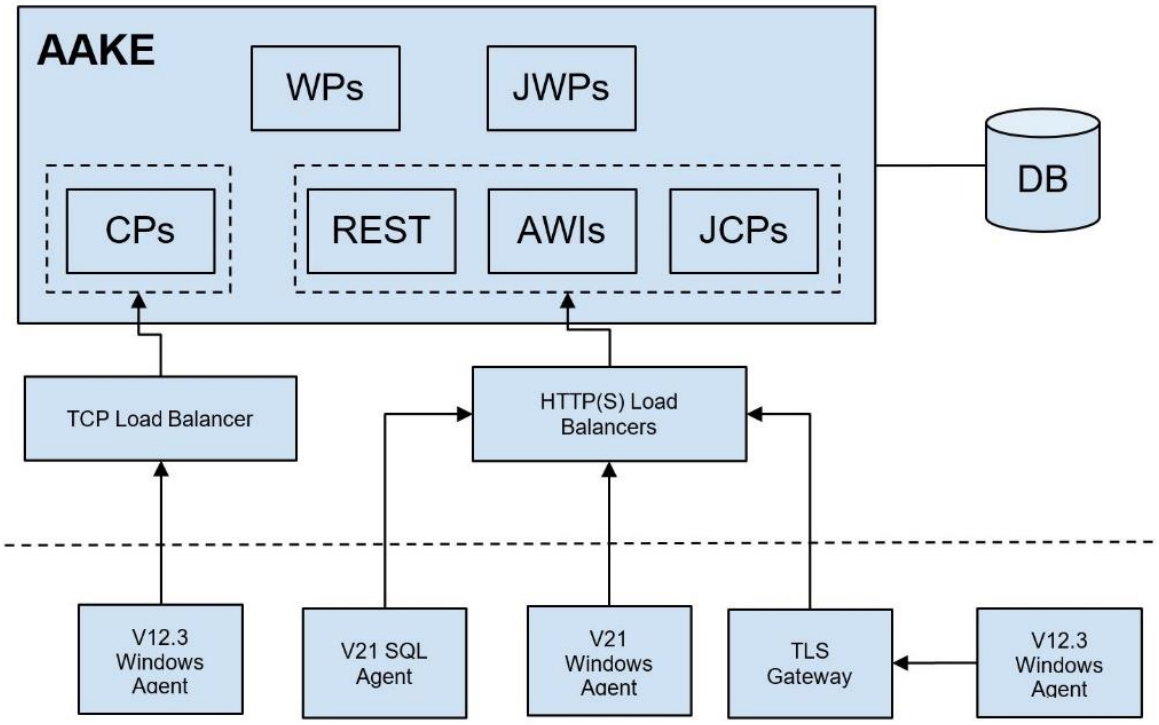
If you are new to Azure and Kubernetes, the AKS quickstart guides can provide you with all the information you need, starting with how to install and configure the Azure CLI, to advanced security and authentication options for your cluster.

Azure prerequisites at a glance:

- AKS Kubernetes Cluster
- PostgreSQL database
- Access to the ingress-nginx Helm repository
- Access to the jetstack Helm repositories

Be aware, this document does not replace the Automic documentation or a basic understanding of Kubernetes concepts and other Cloud relevant components, such as Load Balancers.

The following is only an example of how to deploy AAKE on AKS. There are multiple options with the many services Azure provides, some of which might better fit your needs. In the scenario described in this guide HTTP(s) and TCP Load balancers are used to allow both TLS and non-TLS agents to connect to the cluster. An overview can be found in the diagram below.



.

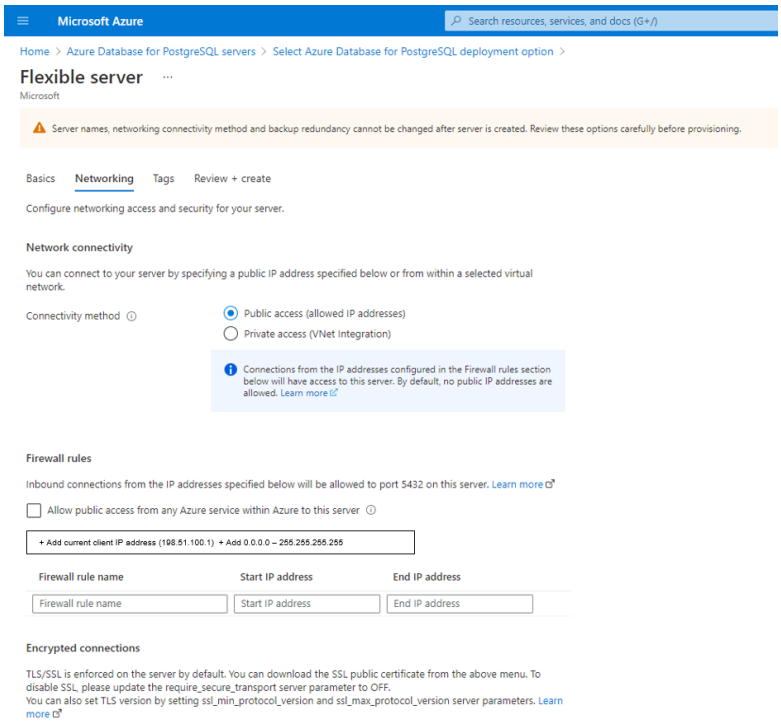
Chapter 2: Create the PostgreSQL Database for Automic

The Azure Database for PostgreSQL provides a fully managed DB service that Automic Automation V21 supports.

A TLS/SSL secure database connection is currently not supported by Automic Automation Kubernetes edition. As TLS/SSL is enforced on the Azure PostgreSQL server by default, you must disable it by updating the require_secure_transport server parameter to OFF.

There are some mandatory settings to optimize the database and connections for the Automic Server. You will need to create a new parameter group and set vacuum_cost_limit to 10000 and client_encoding to LATIN9.

The Server parameters can be set after the creation of the DB instance via the Azure portal, or via the CLI.



Once the instance is available, you can connect to it and create a new database:

```
$ psql --host=aake-postgresql.postgres.database.azure.com --port=5432 \
      --username=automic --password --dbname=postgres

  postgres=> CREATE DATABASE ae WITH OWNER = "automic" TEMPLATE = template0 ENCODING
= 'UTF8'
            LC_COLLATE = 'C' LC_CTYPE = 'C' CONNECTION LIMIT = -1;

  postgres=> \c ae

  ae=> CREATE SCHEMA dbo AUTHORIZATION "automic";

  ae=> ALTER ROLE "automic" IN DATABASE ae SET search_path TO 'dbo';
```

Note: No additional tablespaces were created in this example, so the PostgreSQL default of pg_default will be used.

Chapter 3: Create the Secrets

Sensitive information relevant to the Automic system is stored in secrets and retrieved during deployment. You must download a json file that stores the credentials required to pull the container images from the Automic Downloads Page.

3.1 Automic ImagePullSecret used to retrieve the images from GCR

```
$ kubectl create secret docker-registry automic-image-pull-secret \
  --docker-server=gcr.io \
  --docker-username=_json_key \
  --docker-password="$(cat ./automic-image-pull-secret.json)" \
  --docker-email=broadcom-com@esd-automic-saas.iam.gserviceaccount.com
```

3.2 DB secret with the connection information

```
$ kubectl create secret generic ae-db \
--from-literal=host=aake-postgresql.postgres.database.azure.com \
--from-literal=vendor=postgres \
--from-literal=port='5432' \
--from-literal=user=automic \
--from-literal=db=ae \
--from-literal=password=automic \
--from-literal=data-tablespace-name=pg_default \
--from-literal=index-tablespace-name=pg_default \
--from-literal=additional-parameters="connect_timeout=10 client_encoding=LATIN9"
```

3.3 Client 0 secret with pre-set credentials

```
$ kubectl create secret generic client0-user \
--from-literal=client='0' \
--from-literal=user=ADMIN \
--from-literal=department=ADMIN \
--from-literal=password=admin
```

Chapter 4: Create a Public IP Address

A typical scenario assigns an existing static public IP address to the NGINX ingress controller.

First, you will need the resource group of your Kubernetes cluster

```
$ az aks show --resource-group sandbox \
--name aake-demo \
--query nodeResourceGroup -o tsv
```

Where sandbox is the overall resource group, and aake-demo the name of the cluster.

Example value MC_sandbox_aake-demo_eastus

To create a new public IP address:

```
$ az network public-ip create \
--resource-group MC_sandbox_aake-demo_eastus \
--name aake-demo-public-ip \
--sku Standard \
--allocation-method static \
--query publicIp.ipAddress -o tsv
```

Record the IP address created.

NOTE Be aware that this public IP address gets deleted when deleting the cluster as it's in the same resource group.

Chapter 5: Create an Ingress Controller

Create a new namespace ingress-basic

```
$ kubectl create namespace ingress-basic
```

Deploy NGINX

```
$ helm install nginx-ingress ingress-nginx/ingress-nginx \
--namespace ingress-basic \
--set controller.replicaCount=2 \
--set controller.nodeSelector."kubernetes\.io/os"=linux \
--set defaultBackend.nodeSelector."kubernetes\.io/os"=linux \
--set controller.admissionWebhooks.patch.nodeSelector."beta\.kubernetes\.io/os"=linux
\
--set controller.service.loadBalancerIP="<your public ip address>" \
--set controller.service.annotations."service\.beta\.kubernetes\.io/azure-dns-label-
name"="automic-aake-demo-dns-label"
```

Where <your public ip address> should match the public IP that was created earlier.

Important: The azure-dns-label-name must be unique within the Azure region.

Check ingress controller status

```
$ kubectl --namespace ingress-basic get services -o wide -w nginx-ingress-ingress-
nginx-controller
```

Make sure the public IP address is listed under EXTERNAL-IP

Chapter 6: Configure TLS certificates

You must have valid certificates in place to connect TLS-enabled agents to the AAKE cluster. The TLS handshake is performed between the agent and the Ingress/Load Balancer and there is no need to additionally configure the JCP as is the case for on-prem installations.

For this purpose, a private/public keypair and a certificate have to be created and configured as a TLS secret in the Ingress. The TLS-enabled agents use hostname verification, so make sure the domains of the HTTP(s) Load Balancers are included as SANs in the certificate if you don't use a wildcard domain.

For example, we use cert-manager, which provides automatic Let's Encrypt certificate generation and management functionality.

Label the ingress-basic namespace to disable cert-manager resource validation

```
$ kubectl label namespace ingress-basic cert-manager.io/disable-validation=true
```

Install the cert-manager Helm chart

```
$ helm install \
cert-manager \
--namespace ingress-basic \
--version v1.5.4 \
--set installCRDs=true \
--set nodeSelector."kubernetes\.io/os"=linux \
jetstack/cert-manager
```

Create a CA cluster issuer, cluster-issuer.yaml

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
name: letsencrypt-prod
spec:
acme:
server: https://acme-v02.api.letsencrypt.org/directory
email: admin@automic-kubernetes.com
privateKeySecretRef:
name: letsencrypt-prod
solvers:
- http01:
ingress:
class: nginx
podTemplate:
spec:
nodeSelector:
"kubernetes.io/os": linux
```

Add to the cluster using

```
$ kubectl apply -f cluster-issuer.yaml --namespace ingress-basic
```

Create a certificate resource, certificate.yaml

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: tls-aks-automic-aake-demo
spec:
  secretName: tls-aks-automic-aake-demo-secret
  dnsNames:
  - awi-default.<your public ip address>.nip.io
  - jcp-ws-default.<your public ip address>.nip.io
  - jcp-rest-default.<your public ip address>.nip.io
  acme:
    config:
    - http01:
        ingressClass: nginx
      domains:
      - awi-default.<your public ip address>.nip.io
      - jcp-ws-default.<your public ip address>.nip.io
      - jcp-rest-default.<your public ip address>.nip.io
  issuerRef:
    name: letsencrypt-prod
    kind: ClusterIssuer
```

Where <your public ip address> should match the public IP that was created earlier.

Important: the secret name tls-aks-automic-aake-demo-secret will be used in the values.yaml when deploying AAKE.

Submit the cert issue request

```
$ kubectl apply -f certificate.yaml --validate=false
```

Check the certificate status

```
$ kubectl describe certificate tls-aks-automic-aake-demo
```

Chapter 7: Deploy AAKE in the AKS Cluster

The AAKE zip package that can be downloaded from https://downloads.automic.com contains a Helm Plugin mainly used to check the status of the installation and a Helm chart with the values.yaml file as the entry point for the configuration.

7.1 Download the AAKE zip package and install the Automic Helm Plugin and Helm chart

```
$ tar xvf automic-automation-plugin-1.0.0.tgz
$ helm plugin install automic-automation-plugin

$ tar xvf automic-automation-1.0.0.tgz
$ cp automic-automation/values.yaml values.yaml
```

7.2 Configure the Ingress in values.yaml

Since Azure AKS supports the NGINX Controller, the Ingresses generated automatically for AWI, JCP WS and JCP REST can be used during the deployment.

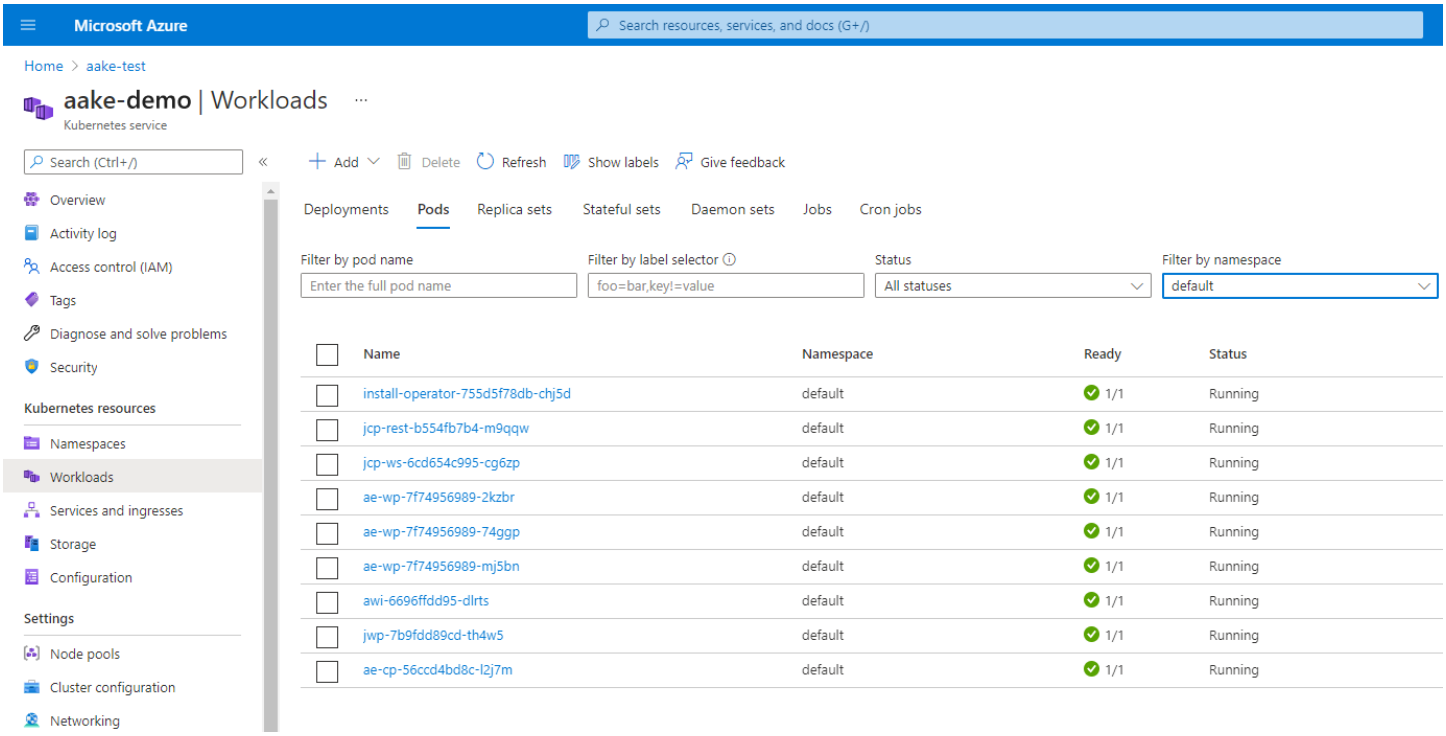
```
# ingress defines the settings for ingress
ingress:
  # enabled creates ingresses for AWI, JCP-WS, JCP-REST and operator if set to true
  enabled: true
  # applicationHostname will be used as suffix for the ingress, e.g. host will be
inside that domain
  applicationHostname:
  # secretName is the name of the TLS secret that contains the key and certificate
to use for the ingress
  secretName: tls-aks-automic-aake-demo-secret
```

7.3 Install AAKE using Helm

```
$ helm install aake automic-automation-1.0.0.tgz -f values.yaml
```

The Automic Helm plugin can be used to check the status of the installation:

```
$ helm automic-automation status
```


Microsoft Azure
Search resources, services, and docs (G+/)
Home > aake-test
aake-demo | Workloads
Kubernetes service
Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Security
Kubernetes resources
Namespaces
Workloads
Services and ingresses
Storage
Configuration
Settings
Node pools
Cluster configuration
Networking
Add
Delete
Refresh
Show labels
Give feedback
Deployments
Pods
Replica sets
Stateful sets
Daemon sets
Jobs
Cron jobs
Filter by pod name
Enter the full pod name
Filter by label selector
foo=bar,key!=value
Status
All statuses
Filter by namespace
default
Name
Namespace
Ready
Status
install-operator-755d5f78db-chj5d default 1/1 Running
jcp-rest-b554fb7b4-m9qqw default 1/1 Running
jcp-ws-6cd654c995-cg6zp default 1/1 Running
ae-wp-7f74956989-2kzbr default 1/1 Running
ae-wp-7f74956989-74ggp default 1/1 Running
ae-wp-7f74956989-mj5bn default 1/1 Running
awi-6696ffdd95-dlrts default 1/1 Running
jwp-7b9fdd89cd-th4w5 default 1/1 Running
ae-cp-56ccd4bd8c-l2j7m default 1/1 Running

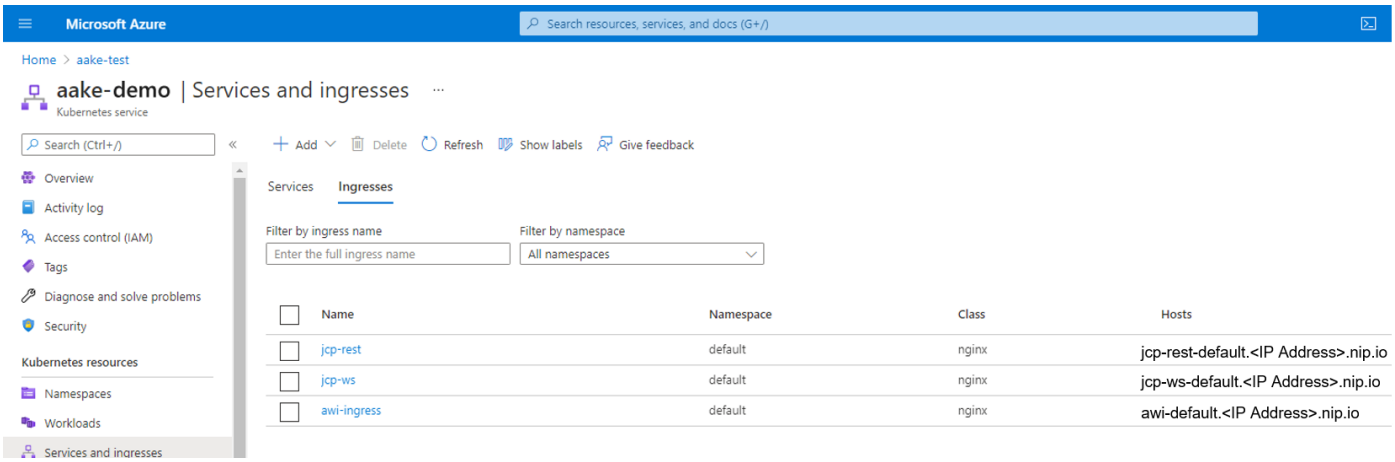
Chapter 8: Expose the Cluster to the outside world

8.1 Expose the AWI, JCP WS and JCP REST services

To access the AWI and for TLS agents/Gateway to connect to the JCP, you have to expose the cluster services via Ingresses and HTTP(S) Load Balancers.

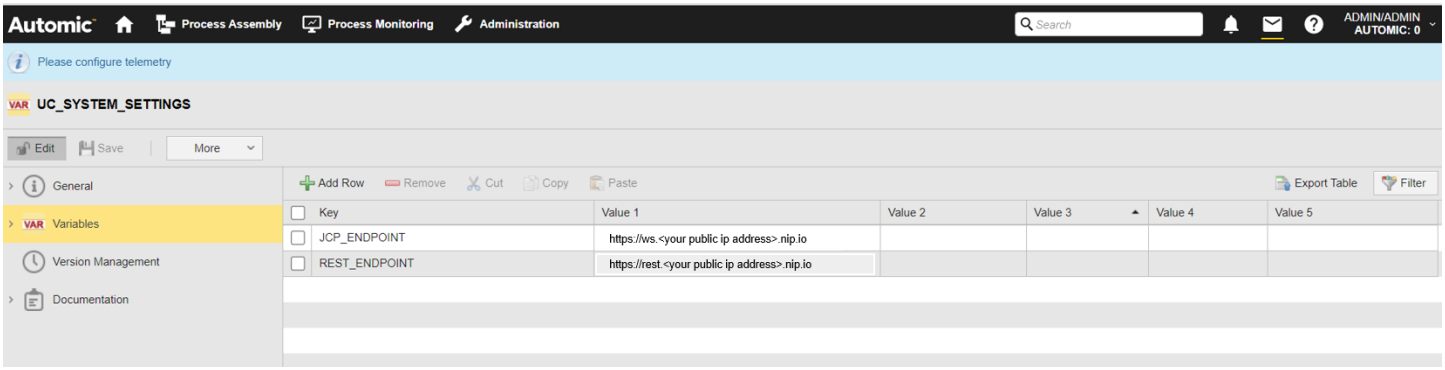
The domains/endpoints for the HTTP(S) Load Balancers are configured as hosts in the Ingresses, and the TLS certificate is referenced via the configured TLS secret.

The Ingresses automatically generated during the deployment are configured for an NGINX Controller and can be used to access AWI, JCP-REST and JCP-WS.



After the Ingresses have been successfully deployed and Load Balancers created, AWI can be reached via the exposed endpoint https://awi-default.<your public ip address>.nip.io.

Additionally, the endpoints for the WS and REST JCPs need to be configured in UC_SYSTEM_SETTINGS to also point to the Load Balancer address.



If you already have domains/addresses assigned to the Load Balancer(s), you can also configure the endpoints as environment variables in values.yaml.

```
# environment defines variables that will be stored in the configmap aa-properties
and injected as ENV into the containers environment:
  JCP_WS_EXTERNAL_ENDPOINT: "https://ws-default.<your public ip address>.nip.io"
  JCP_REST_EXTERNAL_ENDPOINT: "https://rest-default.<your public ip address>.nip.io"
```

8.2 Expose the CP Services

When HTTP(S) Load Balancers can't be used to connect non-TLS agents or when OS CallAPIs are required, you will need to expose the CPs within the cluster via the TCP Load Balancers. Note that by default, the values.yaml does not instantiate any CP pods (cpReplicas: 0)

You do this by creating a Kubernetes Service of type Load Balancer, cp-lbsvc.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: cp-loadbalancer
spec:
  selector:
    app.kubernetes.io/instance: cp-automic-automation
    app.kubernetes.io/name: cp
  ports:
    - protocol: TCP
      port: 2217
      targetPort: 2217
  type: LoadBalancer
```

Create the load balancer service

```
$ kubectl apply -f cp-lbsvc.yaml
```

The Load Balancer will be automatically created and configured with a new public IP address, that can be used by the non-TLS agents and CallAPIs to connect to the CPs.. To check the load balancer and view the external IP address, issue

```
$ kubectl get service cp-loadbalancer
```

The endpoints for the CPs need to be configured in AWI, Client 0, UC_SYSTEM_SETTINGS to point to the TCP Load Balancer address.

If you a have a public address assigned to the LB before the AAKE deployment, you can also configure the endpoint as environment variables in values.yaml.

```
# environment defines variables that will be stored in the configmap aa-properties
and injected as ENV into the containers environment:
  CP_EXTERNAL_ENDPOINT: "<your tcp load balancer external ip address>"
```

Chapter 9: Connect Agents via TCP or HTTPS Load Balancer

9.1 Connect TLS-enabled agents

TLS agents can connect to the JCPs via the Ingress/HTTPS Load Balancer that acts as a server for the TLS Handshake. The certificate of the LB needs to be trusted by the agent. Since a public CA signs it, it is usually trusted by applications since the root certificate of the signing CA is already included in the Java/OS truststore.

In this case, the ini file of the v21 Windows agent and TLS Gateway only require the Automic system name and endpoint where the JCPs can be reached:

UCXJWX6.ini:

```
[GLOBAL]
;
name=WINTLS01
;
system=AUTOMIC
    ...
[TCP/IP]
;
connection=ws-default.<your public ip address>.nip.io:443
```

ucxjsqlx.ini:

```
[GLOBAL]
;
name=SQLTLS01
;
system=AUTOMIC
…
[TCP/IP]
;
connection=ws-default.<your public ip address>.nip.io:443
```

9.2 Connect non-TLS agents via TLS Gateway

To use the TLS Gateway in CP mode, the TLS_GATEWAY_CP key in UC_SYSTEM_SETTINGS variable has to be set to Yes, and the cp_port ini parameter has to be configured. Configure the required param in the ini file of the Gateways as below:

uctlsgtw.ini:

```
[GLOBAL]
;
name=TLSGTW01
;
system=AUTOMIC
…
[TCP/IP]
;
connection=ws-default.<your public ip address>.nip.io:443
…
cp_port=2217
```

The v12.3 agents can use the same system name and the cp parameter has to match the hostname/address of the machine where the TLS Gateway is installed and also the same port configured as a cp_port for the Gateway.

For our example, this would be:

UCXJWX6.ini:

```
[GLOBAL]
;
name=WIN12.3GTW
;
system=AUTOMIC
…
[TCP/IP]
;
cp=<your tls gateway hostname or address>:2217
```

9.3 Connect non-TLS agents via TCP Load Balancer

In this case, the 12.3 agent connects directly to the TCP Load Balancer that will forward the requests to the CPs within the cluster.

UCXJWX6.ini:

```
[GLOBAL]
;
name=WIN12.3CP
;
system=AUTOMIC
…
[TCP/IP]
;
cp=<your tcp load balancer external ip address>:2217
```


BROADCOM®