Installing Automic Automation v21 Containerized Agent

How to build and run a container hosting an Automic Unix / Linux Agent

Version 1.1

Broadcom reserves the right to make changes without further notice to any products or data herein to improve reliability, function, or design. Information furnished by Broadcom is believed to be accurate and reliable. However, Broadcom does not assume any liability arising out of the application or use of this information, nor the application or use of any product or circuit described herein, neither does it convey any license under its patent rights nor the rights of others.

Contents

Chapter 1: Introduction

This document enables you to set up and configure an Automic Automation Linux agent running in a Docker or Kubernetes environment.

1.1 Prerequisites

- Automic Automation V21 (preferably 21.0.4), either standard deployment or K8s deployment
- Administrative access to Automic Automation and K8s cluster

.

Chapter 2: Build Time

You must provide all the resources the agent needs during runtime at build time. This includes the following at a minimum:

- Docker base image
- Agent binaries (minimum 21.0.4)
- Ini files.

An image should be generic so you can reuse it for several instances. Therefore, we recommend adding only resources that do not change for different instances. It should not include any instance-specific configuration.

Please see the provided Docker file (Dockerfile) for the agent.

```
[1]     FROM ubuntu:20.04
[2]
[3]     RUN mkdir /opt/agent-linux
[4]     COPY ucxjlx6.tar.gz /opt/agent-linux
[5]
[6]     WORKDIR /opt/agent-linux
[7]     RUN tar -xzf ucxjlx6.tar.gz
[8]
[9]     WORKDIR /opt/agent-linux/bin
[10]    COPY --chmod=0755 run.sh .
[11]
[12]    ENTRYPOINT [ "/opt/agent-linux/bin/run.sh" ]
```

Line 1 identifies that a plain vanilla Linux Docker base image is used. The next step creates a directory, and the Agent's binaries as a gz-file are copied into that directory (line 4) and extracted in line 7. Copying the prepared run script into the container and setting the file permissions (line 10). The last line sets the script to be executed when the Docker image starts.

This script (_build.sh) creates the Docker image

```
#!/bin/sh

export DOCKER_BUILDKIT=1
docker build --no-cache . -t agent-linux:21.0
```

Next, the Docker image should be published (pushed) to a suitable container registry. Here's an example of Docker tag and push command for use with GCP's Artifact Registry. Change accordingly to match your environment:

```
gcloud auth configure-docker europe-central2-docker.pkg.dev
docker tag agent-linux:21.0 europe-central2-docker.pkg.dev/demos/agents/agent-linux:21.0
docker push europe-central2-docker.pkg.dev/demos/agents/agent-linux:21.0
```

Chapter 3: Runtime

At runtime custom configuration for a specific container (instance of an image) have to be set.

3.1 Persistent Volume Claims (PVC)

Any files that must persist between executions of the agent must reside in persistent volumes. The following YAML file (pvc.yaml) contains the definition.

```
[1]    apiVersion: v1
[2]    kind: PersistentVolumeClaim
[3]    metadata:
[4]      name: agent-linux-security-pvc
[5]    spec:
[6]      accessModes:
[7]        - ReadWriteOnce
[8]      resources:
[9]        requests:
[10]         storage: 10Mi
[11]   ---
[12]   apiVersion: v1
[13]   kind: PersistentVolumeClaim
[14]   metadata:
[15]     name: agent-linux-backup-pvc
[16]   spec:
[17]     accessModes:
[18]     - ReadWriteOnce
[19]     resources:
[20]       requests:
[21]         storage: 10Gi
```

When setting the PVC size for the security files (line 10), you have to consider the number of Agents a file transfer will be performed for as a key is stored for each connection. A higher amount of disk storage is claimed for the backup folder (line 21).

3.2 Agent Configuration

The file named configmap.yaml holds the various settings for the agent.

```
[1]    apiVersion: v1
[2]    kind: ConfigMap
[3]    metadata:
[4]      name: agent-linux-config
[5]    data:
[6]      automic_global_name: agent-linux
[7]      automic_global_system: AUTOMIC
[8]    #  automic_tcpip_connection: ws.1.2.3.4.nip.io:443
[9]      automic_tcpip_connection: jcp-ws:8443
[10]     automic_authorization_keypassword: changeit
[11]     automic_variables_uc_ex_ip_addr: 1.2.3.4
[12]     automic_variables_uc_ex_ip_port: "21000"
[13]     automic_variables_uc_ex_job_md_ip_addr: localhost
```

The Agent's name is specified in line 4 and must match Automation Engine's naming conventions. Line 8 contains the external jcp-ws endpoint for when the Agent connects to an external Automic Automation instance. When deploying the Agent to the same cluster where Automic Automation is running, the internal jcp-ws endpoint can be configured (line 9). Line 11 and 12 specify the Agent's external IP address and port used to contact the Agent when a file transfer is initiated. In Docker or Kubernetes environment the variable automic_variables_uc_ex_job_md_ip_addr must be set to localhost.

3.3 Deployment Definition

The YAML file deployment.yaml holds the required settings for a deployment into a Kubernetes cluster.

```
[1]    apiVersion: apps/v1
[2]    kind: Deployment
[3]    metadata:
[4]      name: agent-linux-depl
[5]      labels:
[6]        app: agent-linux
[7]    spec:
[8]      replicas: 1
[9]      selector:
[10]       matchLabels:
[11]         app: agent-linux
[12]     template:
[13]       metadata:
[14]         labels:
[15]           app: agent-linux
[16]       spec:
[17]         containers:
[18]         - name: agent-linux
[19]           image: europe-central2-docker.pkg.dev/demos/agents/agent-linux:21.0
[20]           imagePullPolicy: IfNotPresent
[21]           envFrom:
[22]             - configMapRef:
[23]                 name: agent-linux-config
[24]           volumeMounts:
[25]             - name: security-volume
[26]               mountPath: /opt/agent-linux/bin/security
[27]             - name: trustedcert-volume
[28]               mountPath: /opt/agent-linux/trustedcert
[29]             - name: backup-volume
[30]               mountPath: /opt/agent-linux/backup
[31]         volumes:
[32]         - name: security-volume
[33]           persistentVolumeClaim:
[34]             claimName: agent-linux-security-pvc
[35]         - name: backup-volume
[36]           persistentVolumeClaim:
[37]             claimName: agent-linux-backup-pvc
[38]         - name: trustedcert-volume
[39]           secret:
[40]             secretName: jcp-ws-certificate
[41]             items:
[42]             - key: certificate
[43]               path: automic-cert.pem
[44]             defaultMode: 420
[45]             type: Directory
```

The Agent’s parameters are specified in config map named agent-linux-config (line 23).

Mounting certificates from a path into the pod/container does not work unless you have the path on the node where the Agent runs (and nodes are dynamic). To solve this, create a secret with the cert or use PVs/PVCs and mount the cert into the pod.

Mounting the agentsecurityfolder to an external path only works when the path exists on the node, which is seldom the case, and nodes are very dynamic, so data can get lost. Here PVs and PVCs must be used.

Also, a dedicated PVC is used for the Agent's backup folder.

In a K8s cluster, the certificates are signed by a public CA, but the base image used for the agent container does not include the root certificates in the default paths. Root and intermediate CA certificates must be copied and mounted into the trustedcertsfolder (with a secret or PVCs as above).

3.4 Agent Configuration

The Agent configuration consists of several areas.

- INI file configuration
  The most obvious configurations have to be done in the INI file. We recommend doing this with ENV variables which run.sh sets (see below) in the corresponding settings in the INI file.
  An alternative is to provide the INI file with a volume which is passed at the start of an agent.
- JCP certificate
  The agent has to trust the Automation Engine's JCP. If it's not a public trusted certificate, the certificate has to be added to the trustedCertFolder.
- Agent certificate
  An agent authenticates against the Automation Engine with its security-related files such as private keys, signed certificates and root certificates and stores it in the security folder. This folder must be preserved in a volume. Otherwise, the certificate (and private key) is lost, and the Automation Engine would reject the agent because it thinks it's different.

The script run.sh

```
[1]   #!/bin/sh
[2]
[3]   terminate() {
[4]         echo $0: trapped signal
[5]         if [ ! -z "$PID" ]; then
[6]               echo $0: will kill agent
[7]               kill $PID
[8]         fi
[9]         echo $0: exiting
[10]        exit 0
[11]  }
[12]
[13]  echo $0: will trap SIGINT and SIGTERM
[14]  trap 'kill ${!}; terminate' INT TERM
[15]
[16]  echo "1. adopt agent configuration"
[17]  mv ucxjxxx.ori.ini ucxjlx6.ini
[18]  sed -i "s/^name.*=.*/name=$automic_global_name/g" ucxjlx6.ini
[19]  sed -i "s/^system.*=.*/system=$automic_global_system/g" ucxjlx6.ini
[20]  sed -i "s/^MsgToStdout.*=.*/MsgToStdout=$automic_misc_msgtostdout/g"
   ucxjlx6.ini
[21]  sed -i "s/^connection.*=.*/connection=$automic_tcpip_connection/g"
   ucxjlx6.ini
[22]  sed -i "s/^keyPassword.*=.*/keyPassword=$automic_authorization_keypassword/g"
   ucxjlx6.ini
[23]
[24]  sed -i "s#^trustedCertFolder.*=.*#trustedCertFolder=../trustedcert#g"
   ucxjlx6.ini
[25]  sed -i "s#^initialPackage.*=.*#initialPackage=./package#g" ucxjlx6.ini
[26]
[27]  sed -i "/^\[VARIABLES\].*/a UC_EX_IP_ADDR=$automic_variables_uc_ex_ip_addr"
   ucxjlx6.ini
[28]  sed -i "/^\[VARIABLES\].*/a UC_EX_IP_PORT=$automic_variables_uc_ex_ip_port"
   ucxjlx6.ini
[29]  sed -i "/^\[VARIABLES\].*/a
   UC_EX_JOB_MD_IP_ADDR=$automic_variables_uc_ex_job_md_ip_addr" ucxjlx6.ini
[30]
[31]  echo "2. create user"
[32]  useradd -m agentuser
[33]  echo agentuser:agentpw | chpasswd
[34]
[35]  echo "3. start agent"
[36]  ./ucxjlx6 &
[37]  PID=$!
[38]  wait $PID
```

The script executes when the container starts. In line 17, the Agent's configuration-file template is set to the correct name. Then the parameters in the ini-file are set using sed (lines 18 – 29). After creating an OS user for the Automation Engines JOBS or performing file transfers, the Agent is started (line 36).

3.5 Network Configuration

Hostnames must be resolvable inside the cluster. Either by a configured DNS or services with ExternalName. The setting for UC_EX_JOB_MD_IP_ADDR must be localhost.

3.6 Reset Agent’s Key

When you don't want to store the public/private key in a volume for any reason, you must reset the agent key in the Automation Engine before the agent can connect again. The REST-API can perform this before starting the agent.

Example:

```
curl -f --request POST -u "$CLIENT0_USER/$CLIENT0_DEPARTMENT:$CLIENT0_PASSWORD" \
"http://$REST_CONNECTION/ae/api/v1/0/system/agents/$AGENT_NAME/resetpublickey" -v
```

Credentials for a client 0 user must be provided in addition to the agent’s name and Automation Engine’s REST-endpoint.

Chapter 4: General Advice

4.1 Service Manager

We do not recommend using Service Manager within the container; it increases the container size and complexity without providing any benefits.

Starting and stopping of the agent should be managed by a native container orchestration tool.

4.2 CAU – Centralized Agent Upgrade

It's impossible to use CAU properly because no Service Manager is present. After a container restarts, the agent would reuse the agent image.

Hint: It would be possible to download the latest agent package from the REST-API at every start of the container (see snipped below). This might be suitable for some installations but could cause a considerable load depending on the number of container starts.

```
echo "Create Agent package"
    URL="$REST_ENDPOINT/ae/api/v1/$CLIENT/system/agentpacks"

    BODY="{\"platform\" : \"UNIX\", \"operating_system\" : \"Linux\",
\"operating_system_architecture\" : \"x64\", \"name\" : \"$AGENT_NAME\" }"
    LOCATION=$(curl --header "Content-Type: application/json" \
    --request POST \
    --data "$BODY" \
    --dump-header - \
    --user "$AE_USER_PW" \
    -k \
    "$URL" | grep -i "location" | cut -d" " -f2)

    ZIP_URL="$(sed -e 's/[[:space:]]*$//' <<<${LOCATION})"
    echo "Download $ZIP_URL"
    curl --user "$AE_USER_PW" -k "$ZIP_URL" --output "./tmp.zip"
```

4.3 Starting and Stopping Agents

Any Agent that runs on top of Linux (or Windows) can be started (or stopped) by using the proper administrative view in the Automic Web Interface (AWI).

When using Kubernetes to execute containers, the situation is different because Kubernetes' goal is to keep a container (and therefore the Agent in it) running. So whenever a command stops the Agent from the Automation Engine (AWI), this also ends the execution of the Agent's container. Kubernetes detects that the container is no longer running and restarts it automatically. To prevent the auto-start feature, you must use the Kubernetes' commands to start/stop the Agent's container. (A running container is called a pod).

BROADCOM®