**Run Azure Container Instances**

## Why use Azure Container Instances?

Azure Container Instances is useful for scenarios that can operate in isolated containers, including simple applications, task automation, and build jobs. Here are some of the benefits:

* **Fast startup**: Launch containers in seconds.
* **Per second billing**: Incur costs only while the container is running.
* **Hypervisor-level security**: Isolate your application as completely as it would be in a VM.
* **Custom sizes**: Specify exact values for CPU cores and memory.
* **Persistent storage**: Mount Azure Files shares directly to a container to retrieve and persist state.
* **Linux and Windows**: Schedule both Windows and Linux containers using the same API.

## Create a container

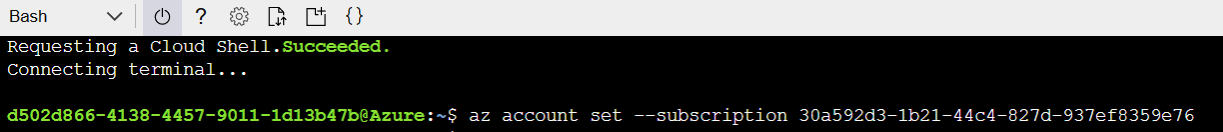
You create a container by providing a name, a Docker image, and an Azure resource group to the az container create command. You can optionally expose the container to the Internet by specifying a DNS name label. In this example, you deploy a container that hosts a small web app. You can also select the location to place the image - you'll use the **East US** region, but you can change it to a location close to you from the following list.

The free sandbox allows you to create resources in a subset of Azure's global regions. Select a region from the following list when creating any resources:

* westus2
* southcentralus
* centralus
* eastus
* westeurope
* southeastasia
* centralindia

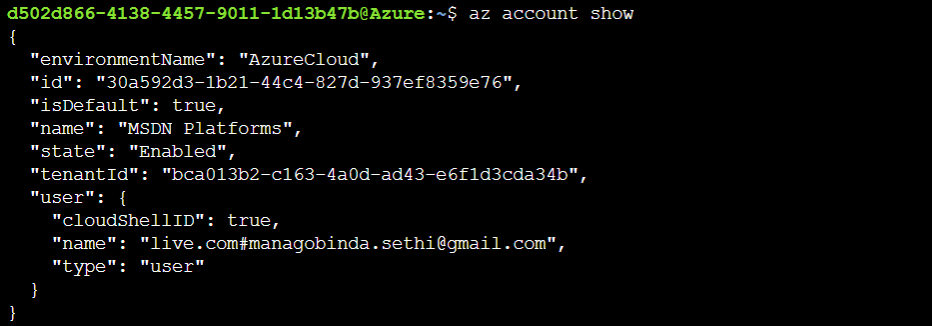
1. Open the cloud shell CLI

We must choose the correct account while connecting to cloud shell

az account set --subscription <subscription ID>

1. Check the account details

az account show



Create a new resource group with the name **containerrg** so that it will be easier to clean up these resources when you are finished with the module. Replace <myLocation> with a region near you.

az group create --name containerrg --location <myLocation>

1. We need to provide a DNS name to expose our container to the Internet. Our DNS name must be unique. For learning purposes, run this command from Cloud Shell to create a Bash variable that holds a unique name.

DNS\_NAME\_LABEL=aci-demo-$RANDOM



1. Create a container with below commands

az container create \

--resource-group containerrg \

--name mycontainer \

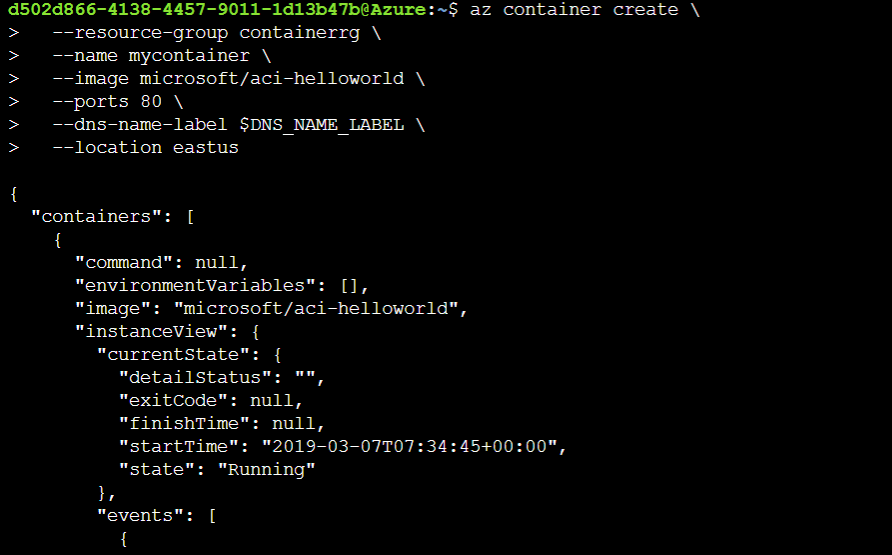
--image mcr.microsoft.com/azuredocs/aci-helloworld \

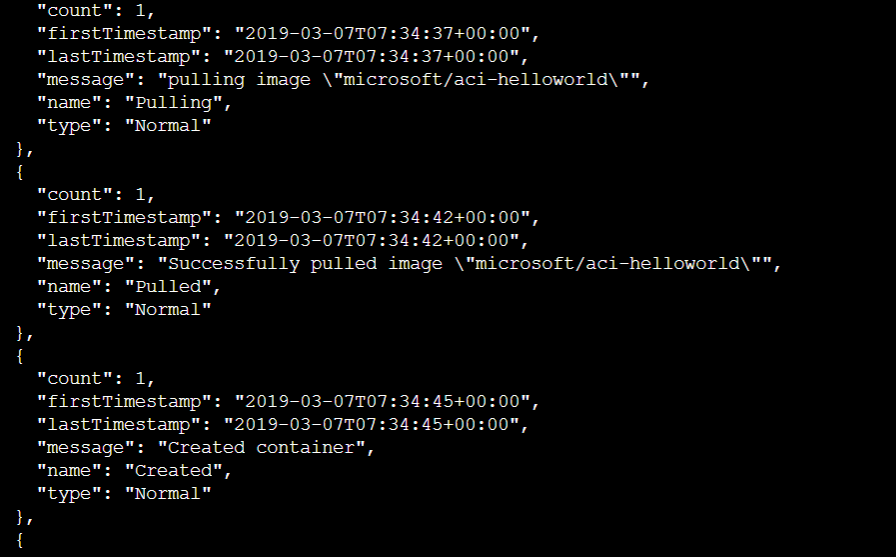
--ports 80 \

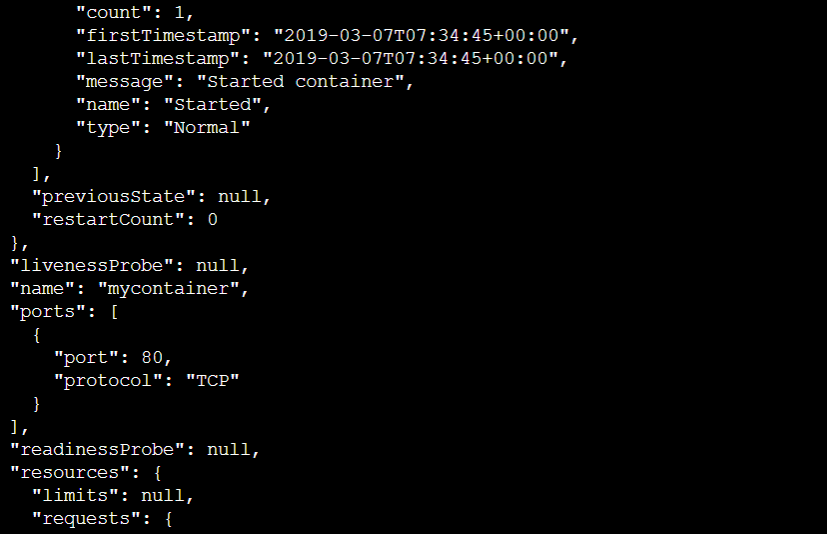
--dns-name-label $DNS\_NAME\_LABEL \

--location eastus

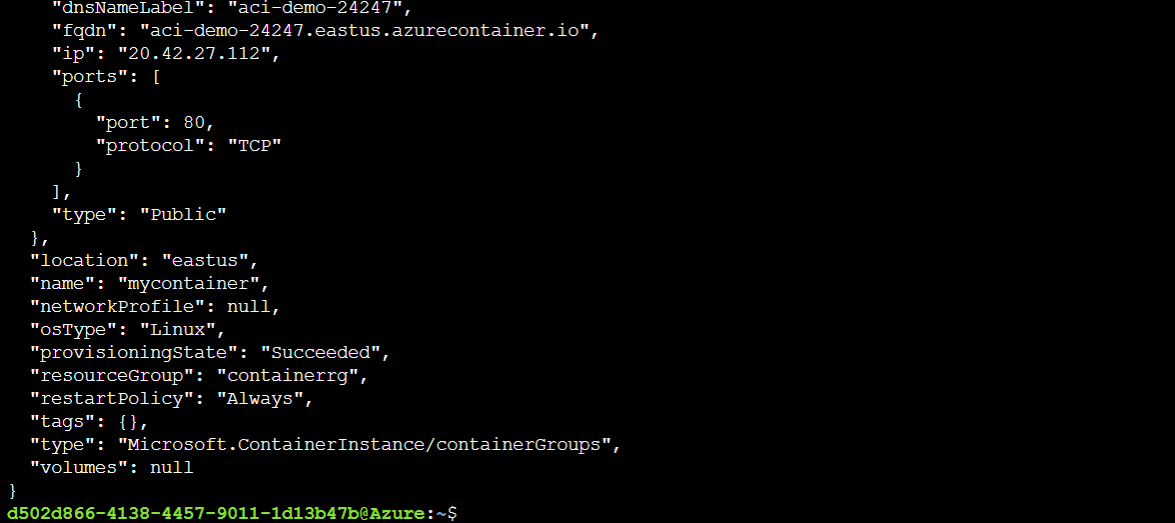
$DNS\_NAME\_LABEL specifies your DNS name. The image name, **mcr.microsoft.com/azuredocs/aci-helloworld**, refers to a Docker image hosted on Docker Hub that runs a basic Node.js web application.











1. When the az container create command completes, run az container show to check its status.

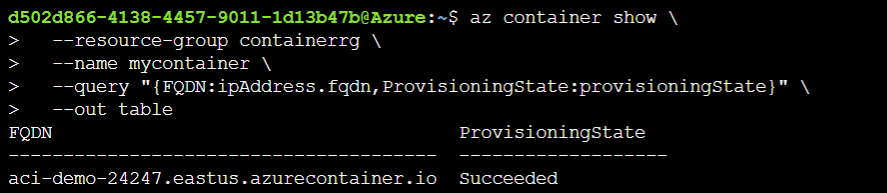
az container show \

--resource-group containerrg \

--name mycontainer \

--query "{FQDN:ipAddress.fqdn,ProvisioningState:provisioningState}" \

--out table



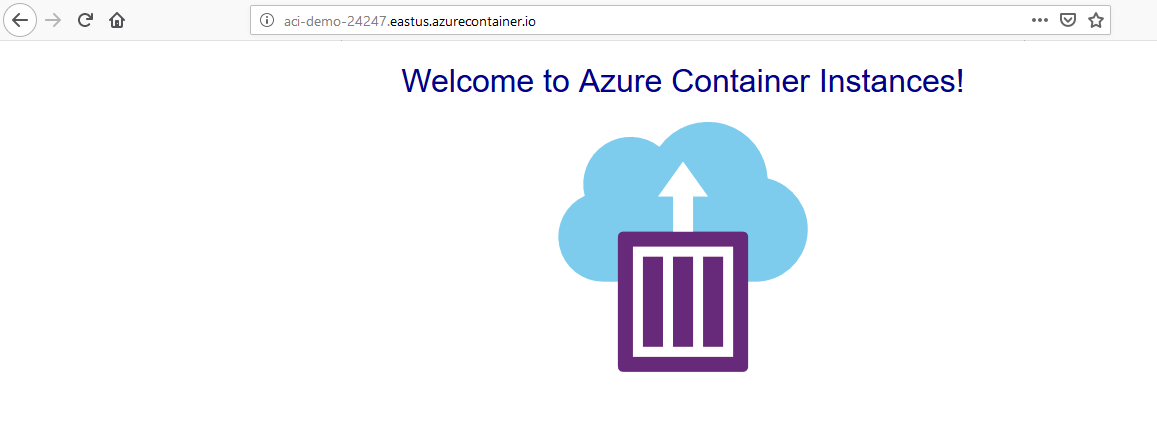
We see our container's fully qualified domain name (FQDN) and its provisioning state. Here's an example.

FQDN ProvisioningState

--------------------------------------- -------------------

aci-demo-24247.eastus.azurecontainer.io Succeeded

1. From a browser, navigate to our container's FQDN to see it running. We see this.



## What are container restart policies?

Azure Container Instances has three restart-policy options:

| **Restart policy** | **Description** |
| --- | --- |
| **Always** | Containers in the container group are always restarted. This policy makes sense for long-running tasks such as a web server. This is the **default** setting applied when no restart policy is specified at container creation. |
| **Never** | Containers in the container group are never restarted. The containers run one time only. |
| **OnFailure** | Containers in the container group are restarted only when the process executed in the container fails (when it terminates with a nonzero exit code). The containers are run at least once. This policy works well for containers that run short-lived tasks. |

**Run a container to completion**

To see the restart policy in action, create a container instance from the **microsoft/aci-wordcount** Docker image and specify the **OnFailure** restart policy. This container runs a Python script that analyzes the text of Shakespeare's Hamlet, writes the 10 most common words to standard output, and then exits.

1. Run this az container create command to start the container.

az container create \

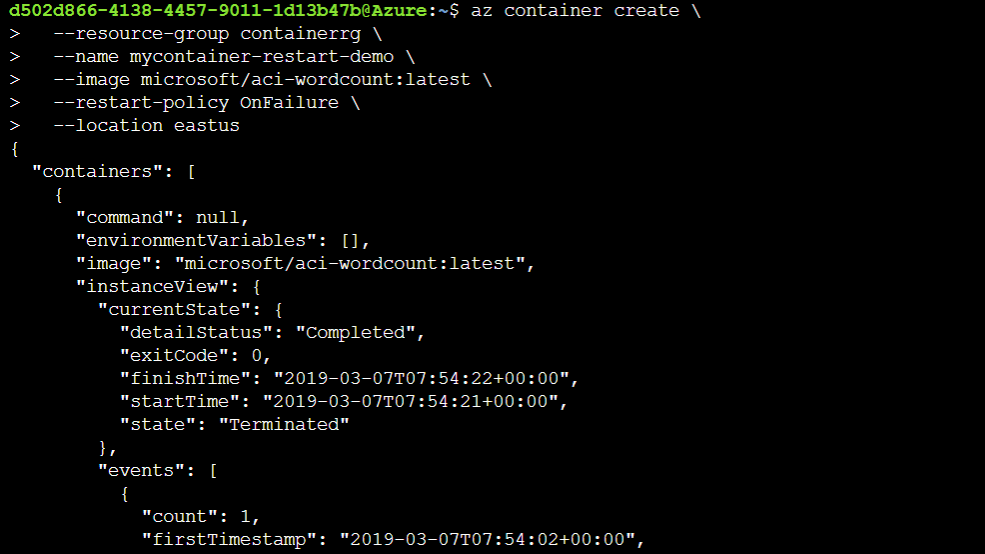
--resource-group containerrg \

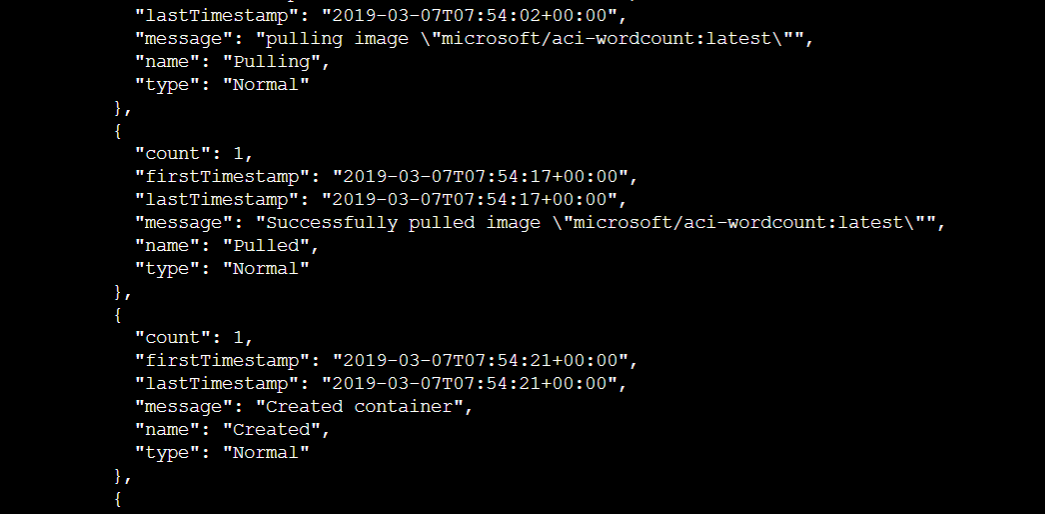
--name mycontainer-restart-demo \

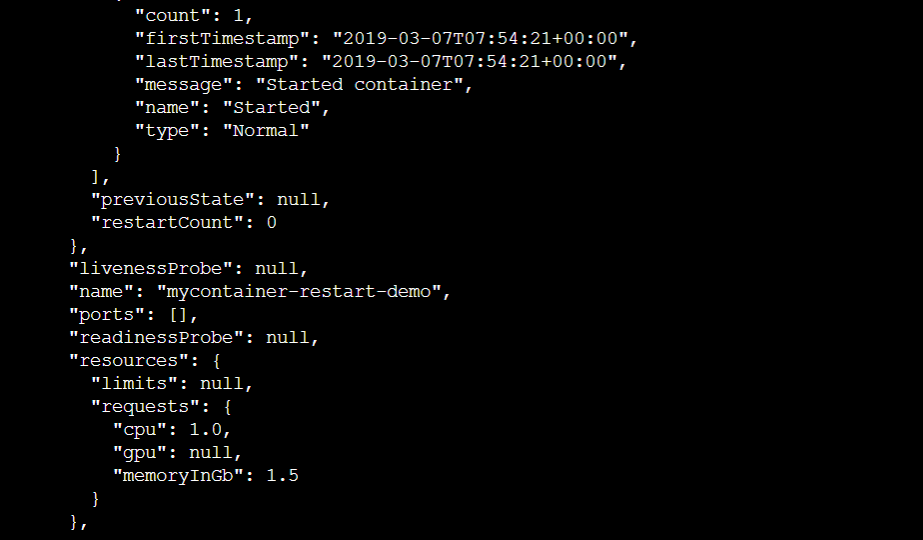
--image microsoft/aci-wordcount:latest \

--restart-policy OnFailure \

--location eastus



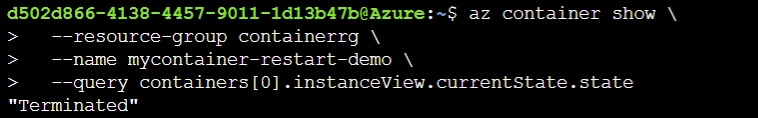




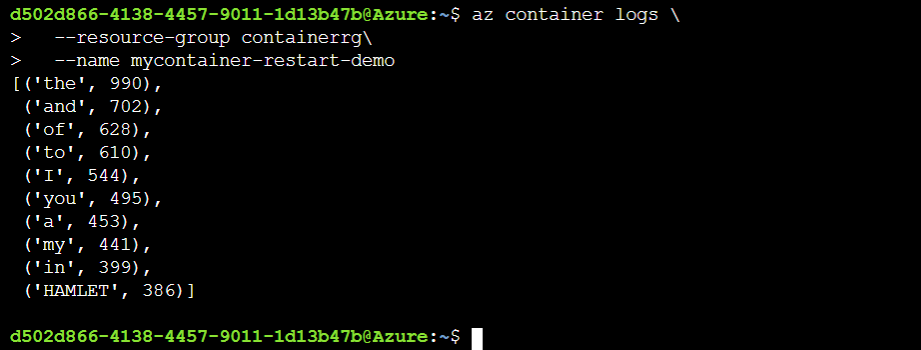


Azure Container Instances starts the container and then stops it when its process (a script, in this case) exits. When Azure Container Instances stops a container whose restart policy is **Never** or **OnFailure**, the container's status is set to **Terminated**.

1. Run az container show to check your container's status and Repeat the command until it reaches the **Terminated** status.



1. View the container's logs to examine the output. To do so, run az container logs like this.



* 1. **Create an Azure container registry**

Create an Azure container registry with the az acr create command. The container registry name must be unique within Azure and contain between 5 and 50 alphanumeric characters.

To begin, we'll define an environment variable in the Cloud Shell called **ACR\_NAME** to hold the name we want to give our new container registry.

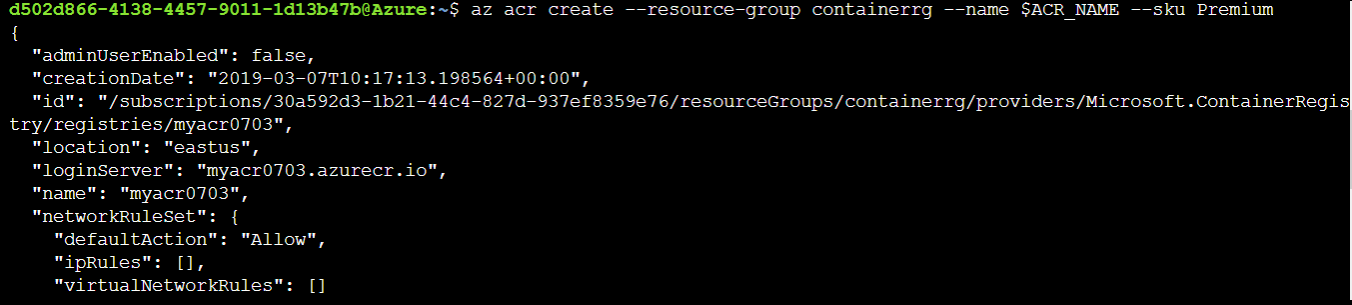
1. Run the following command to define a variable called ACR\_NAME.

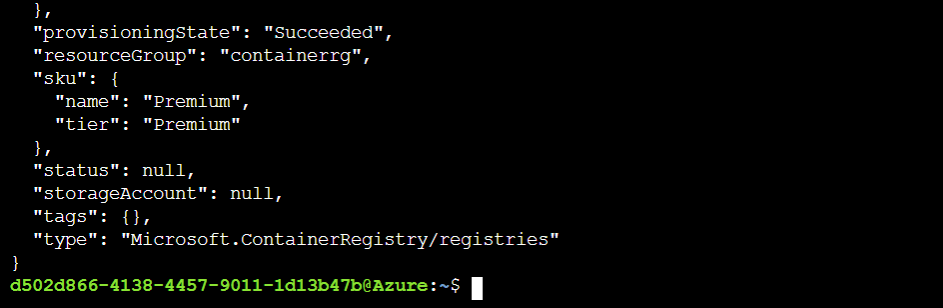
**ACR\_NAME=<registry-name>**



1. Enter the following command into the cloud shell editor to create our new container registry.

**az acr create --resource-group containerrg --name $ACR\_NAME --sku Premium**





## Create a container image with Azure Container Registry Tasks

A standard Dockerfile provides build instructions. Azure Container Registry Tasks allows you to reuse any Dockerfile currently in your environment, including multi-staged builds.

We'll use a new Dockerfile for our example.

Enter the following command into the Cloud Shell window to open the editor.



Copy the following contents into the editor.

**FROM node:9-alpine**

**ADD https://raw.githubusercontent.com/Azure-Samples/acr-build-helloworld-node/master/package.json /**

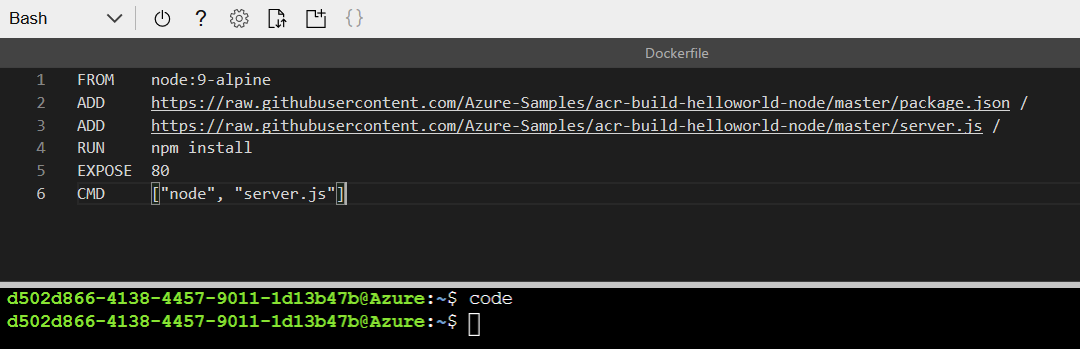
**ADD https://raw.githubusercontent.com/Azure-Samples/acr-build-helloworld-node/master/server.js /**

**RUN npm install**

**EXPOSE 80**

**CMD ["node", "server.js"]**

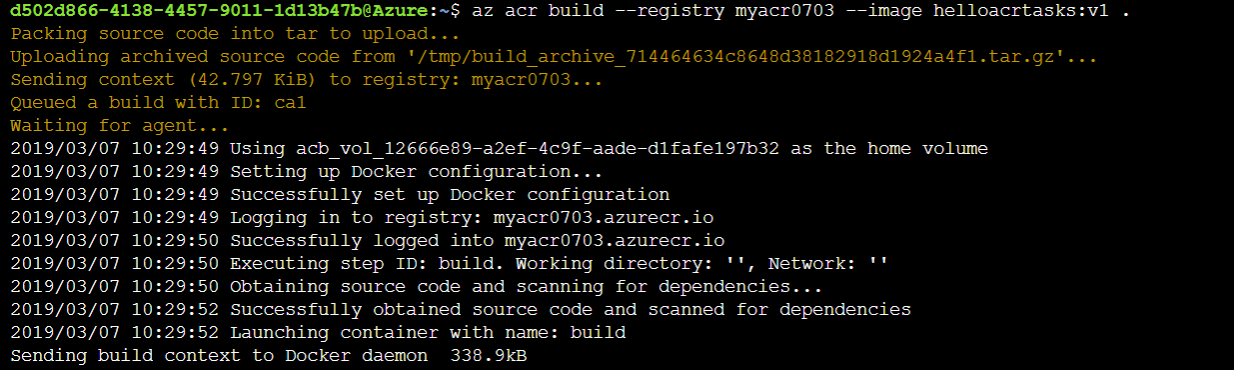
Use the key combination Ctrl+S (Cmd+S for Mac) to save your changes. Name the file as “Dockerfile” when prompted.

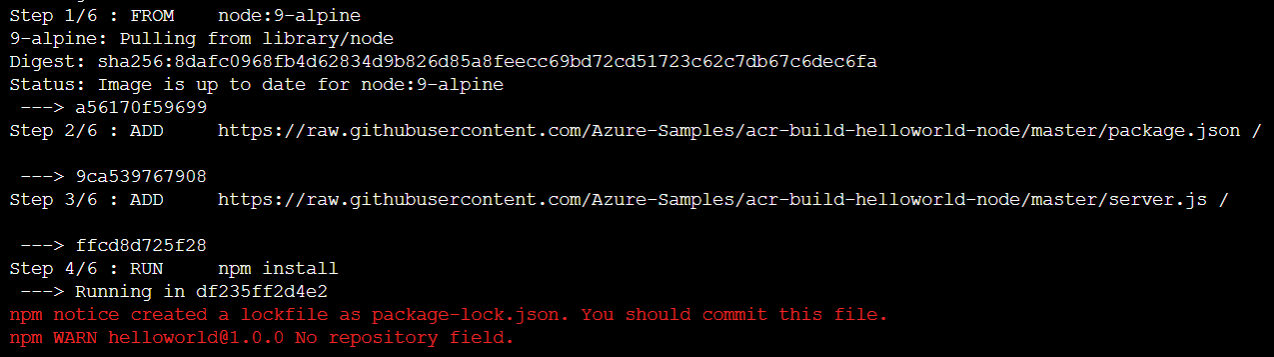


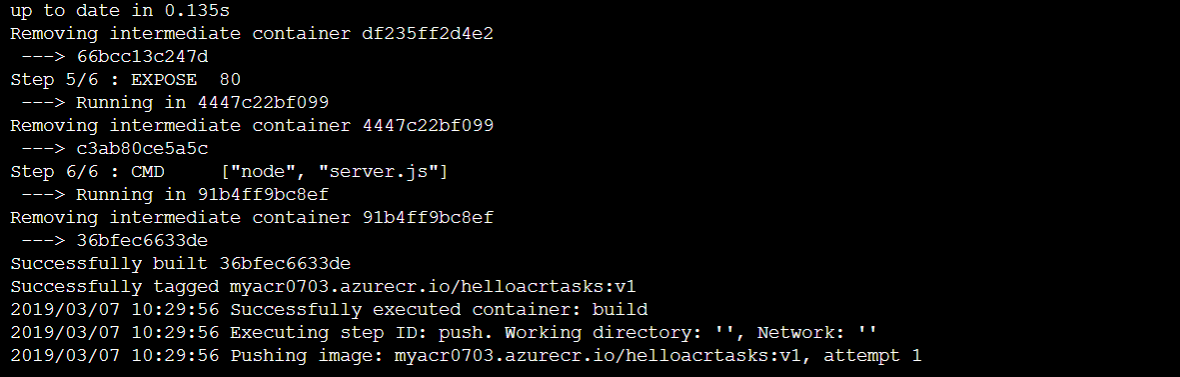
This configuration adds a Node.js application to the node:9-alpine image. After that, it configures the container to serve the application on port 80 via the EXPOSE instruction.

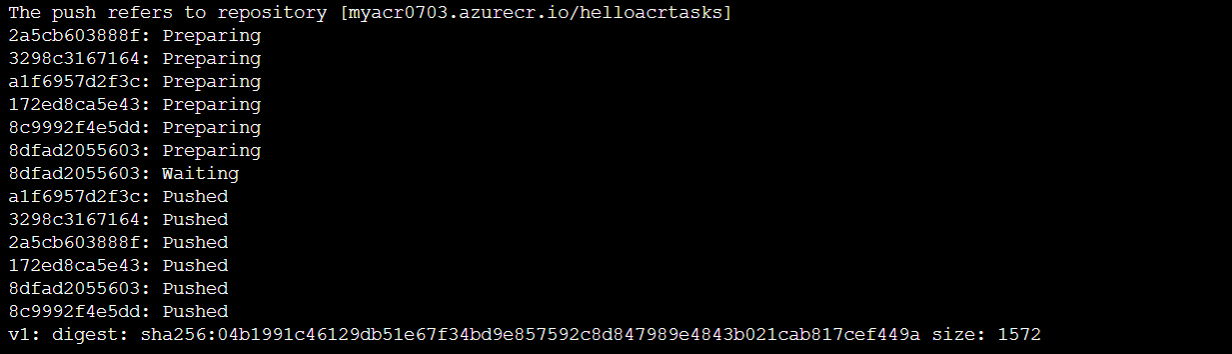
Run the following Azure CLI command to build the container image from the Dockerfile. $ACR\_NAME is the variable you defined in the preceding unit to hold your container registry name.

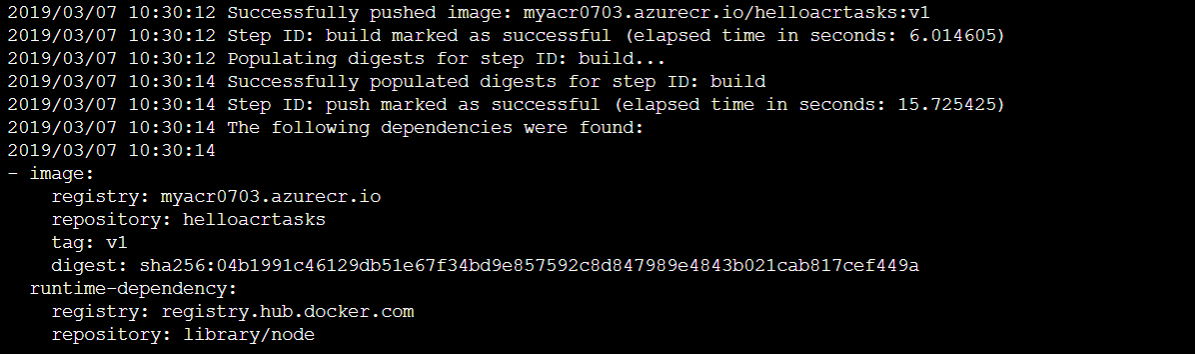
**az acr build --registry $ACR\_NAME --image helloacrtasks:v1 .**

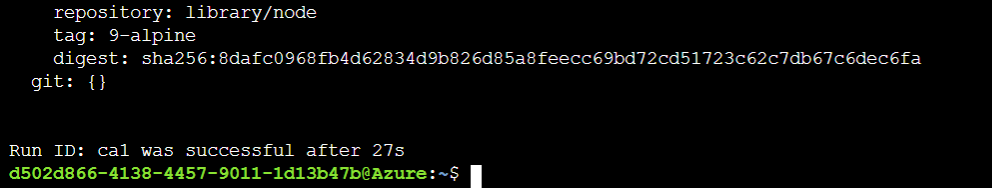








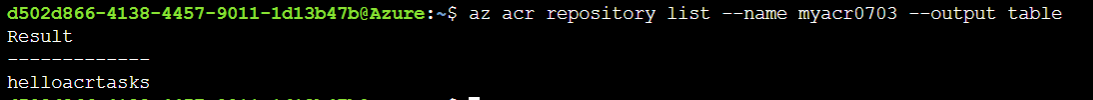




## Verify the image

1. Run the following command in the Cloud Shell to verify that the image has been created and stored in the registry.

**az acr repository list --name $ACR\_NAME --output table**



# Deploy images from Azure Container Registry

First, create a variable in the Cloud Shell named ACR\_NAME with the name of your container registry in lowercase (for example, instead of "MyContainer" make the value "mycontainer"). This variable is used throughout this unit.

**ACR\_NAME=<acrName>**

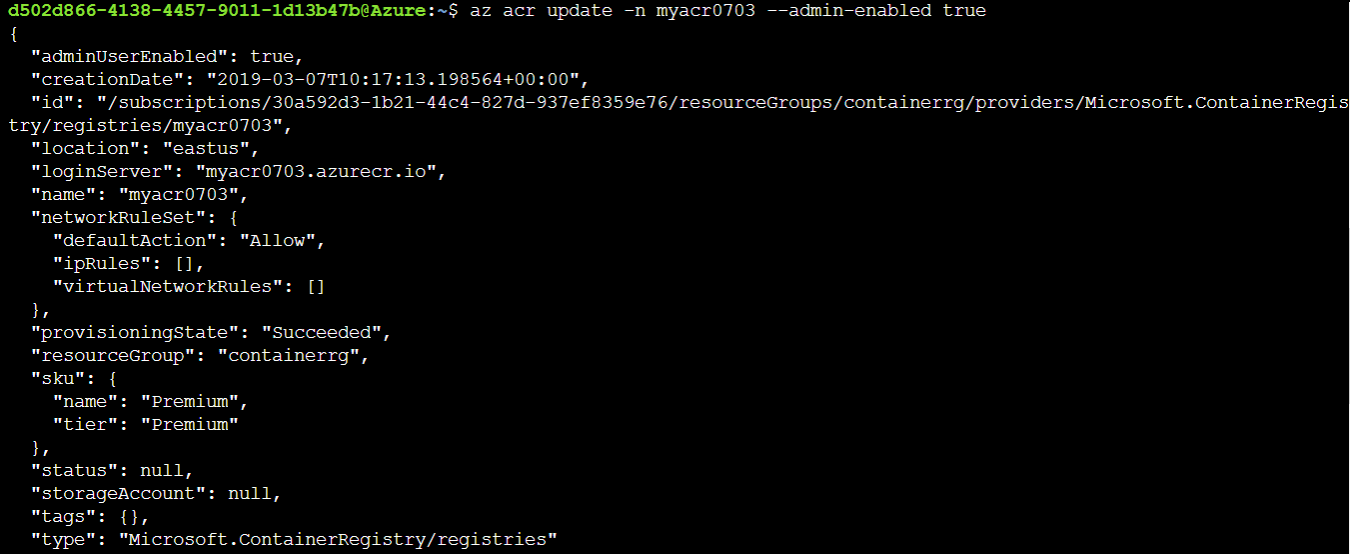


## Enable the registry admin account

In this exercise, we will enable the registry admin account and use it to deploy your image to an Azure Container Instance from the command line.

Run the following commands to enable the admin account on your registry and retrieve its username and password.

**az acr update -n $ACR\_NAME --admin-enabled true**



**az acr credential show --name $ACR\_NAME**

**d502d866-4138-4457-9011-1d13b47b@Azure:~$ az acr credential show --name myacr0703**

**{**

**"passwords": [**

**{**

**"name": "password",**

**"value": "7LBjN+pkIGaHkB9WYxuTU6s6gRYPUJfC"**

**},**

**{**

**"name": "password2",**

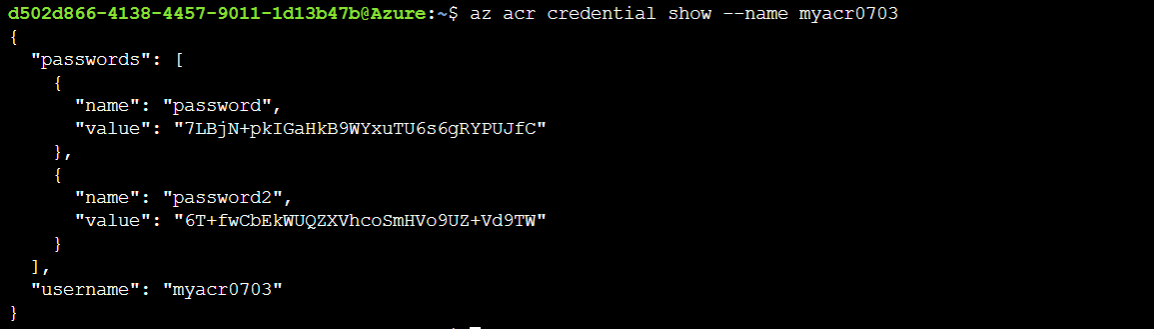
**"value": "6T+fwCbEkWUQZXVhcoSmHVo9UZ+Vd9TW"**

**}**

**],**

**"username": "myacr0703"**

**}**



## Deploy a container with Azure CLI

1. Execute the following az container create command to deploy a container instance. Replace <username> and <password> in the following command with your registry's admin username and password.

**az container create \**

**--resource-group containerrg \**

**--name acr-tasks \**

**--image $ACR\_NAME.azurecr.io/helloacrtasks:v1 \**

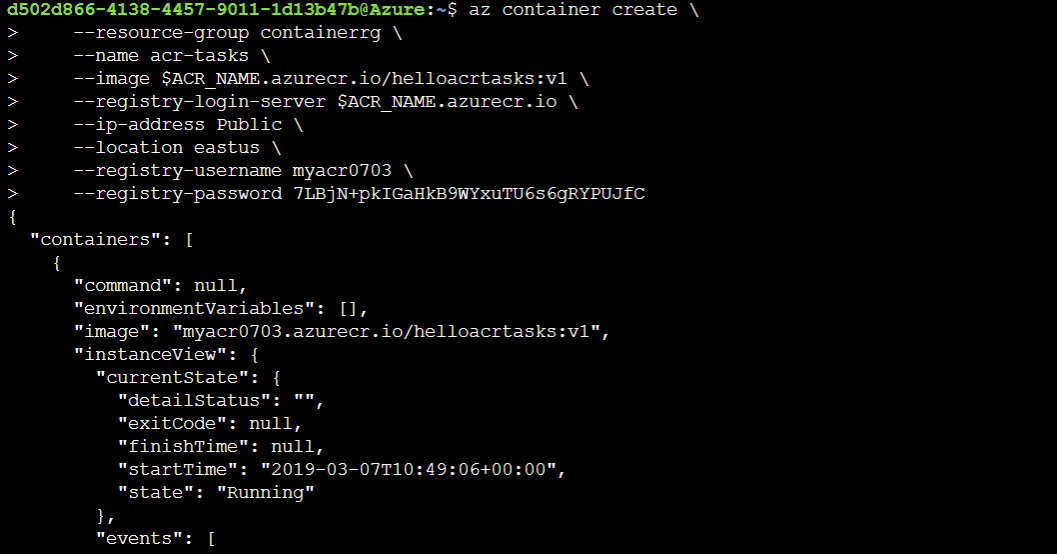
**--registry-login-server $ACR\_NAME.azurecr.io \**

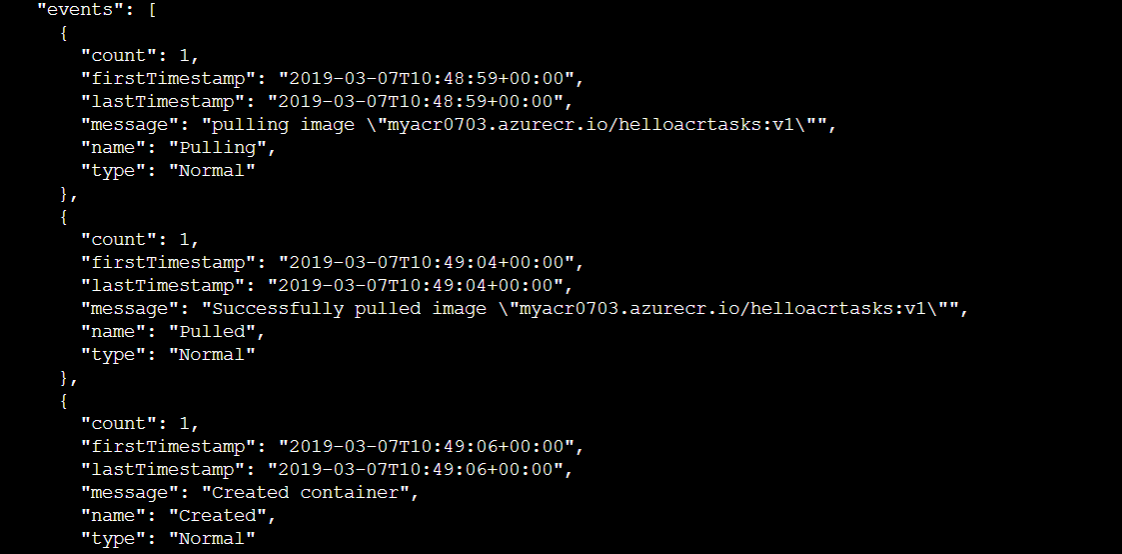
**--ip-address Public \**

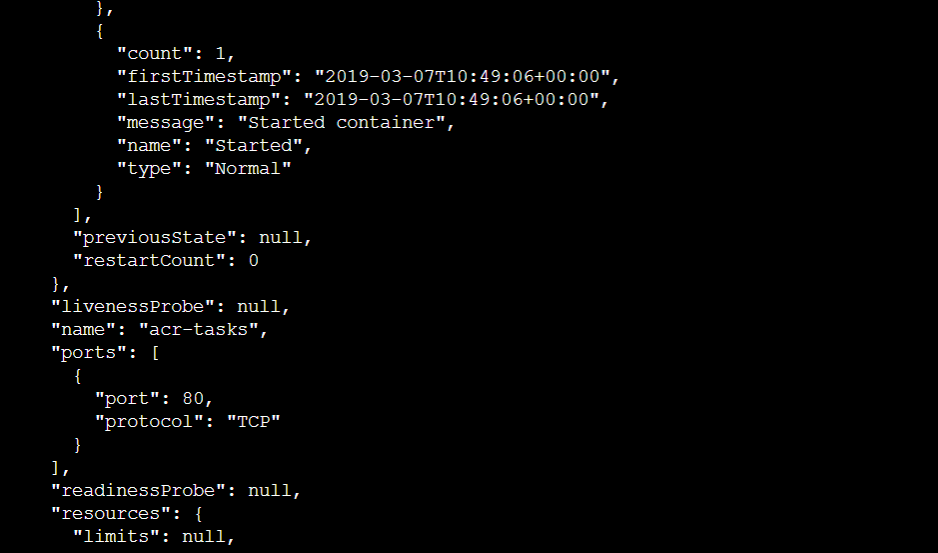
**--location eastus \**

**--registry-username myacr0703 \**

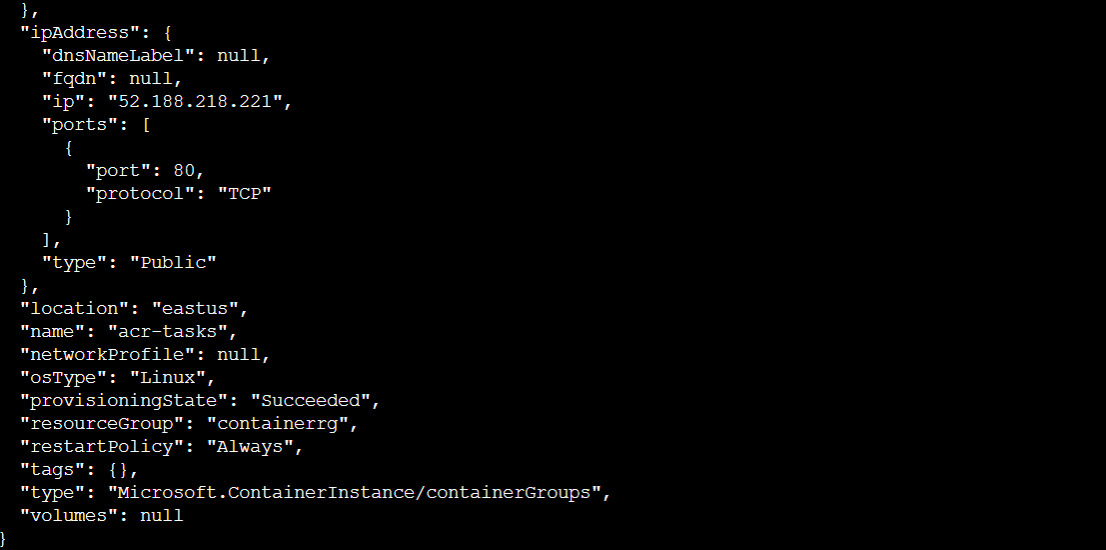
**--registry-password 7LBjN+pkIGaHkB9WYxuTU6s6gRYPUJfC**





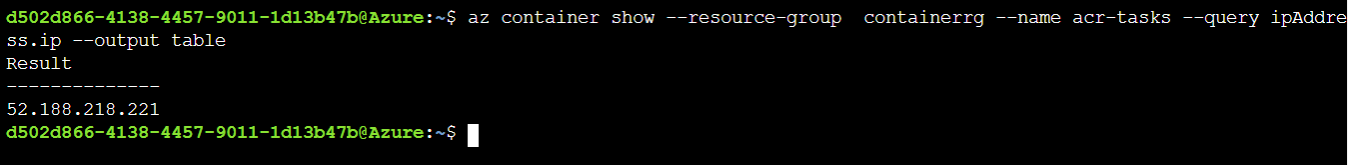


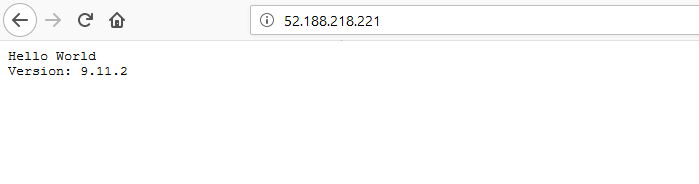




1. Get the IP address of the Azure container instance using the following command.

**az container show --resource-group containerrg --name acr-tasks --query ipAddress.ip --output table**





# Replicate a container image to different Azure regions

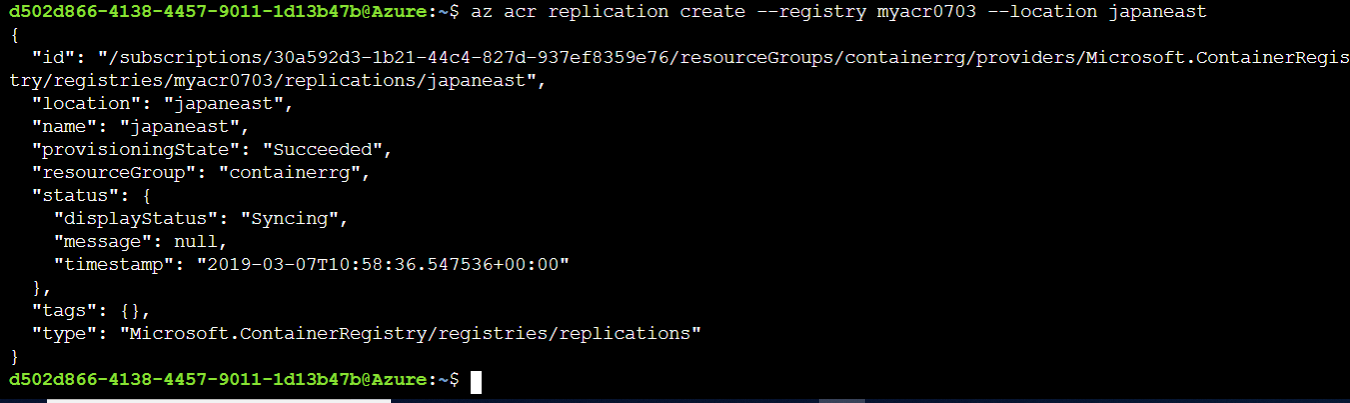
## Replicate a registry to multiple locations

In this exercise, you'll use the az acr replication create Azure CLI command to replicate your registry from one region to another.

1. Run the following command to replicate your registry to another region. In this example, we are replicating to the japaneast region. $ACR\_NAME is the variable you defined earlier in the module to hold your container registry name.

**az acr replication create --registry $ACR\_NAME --location japaneast**

Here's an example of what the output from this command looks like:



1. As a final step, retrieve all container image replicas created by running the following command.

**az acr replication list --registry myacr0703 --output table**

