NEOCEPTION
Pepperl+Fuchs

Neoception® Digital Twin Infrastructure

A Step-by-Step Guide

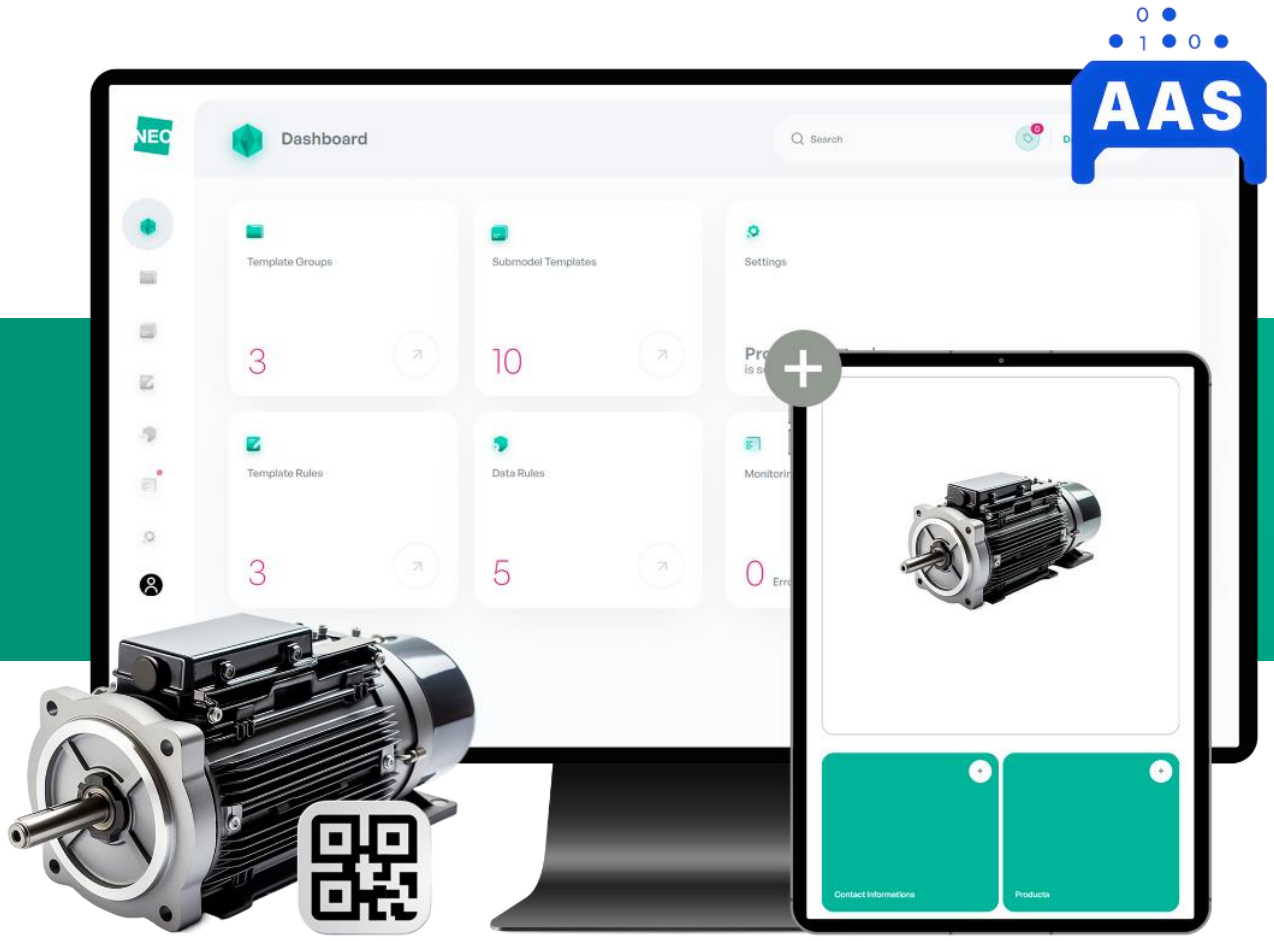


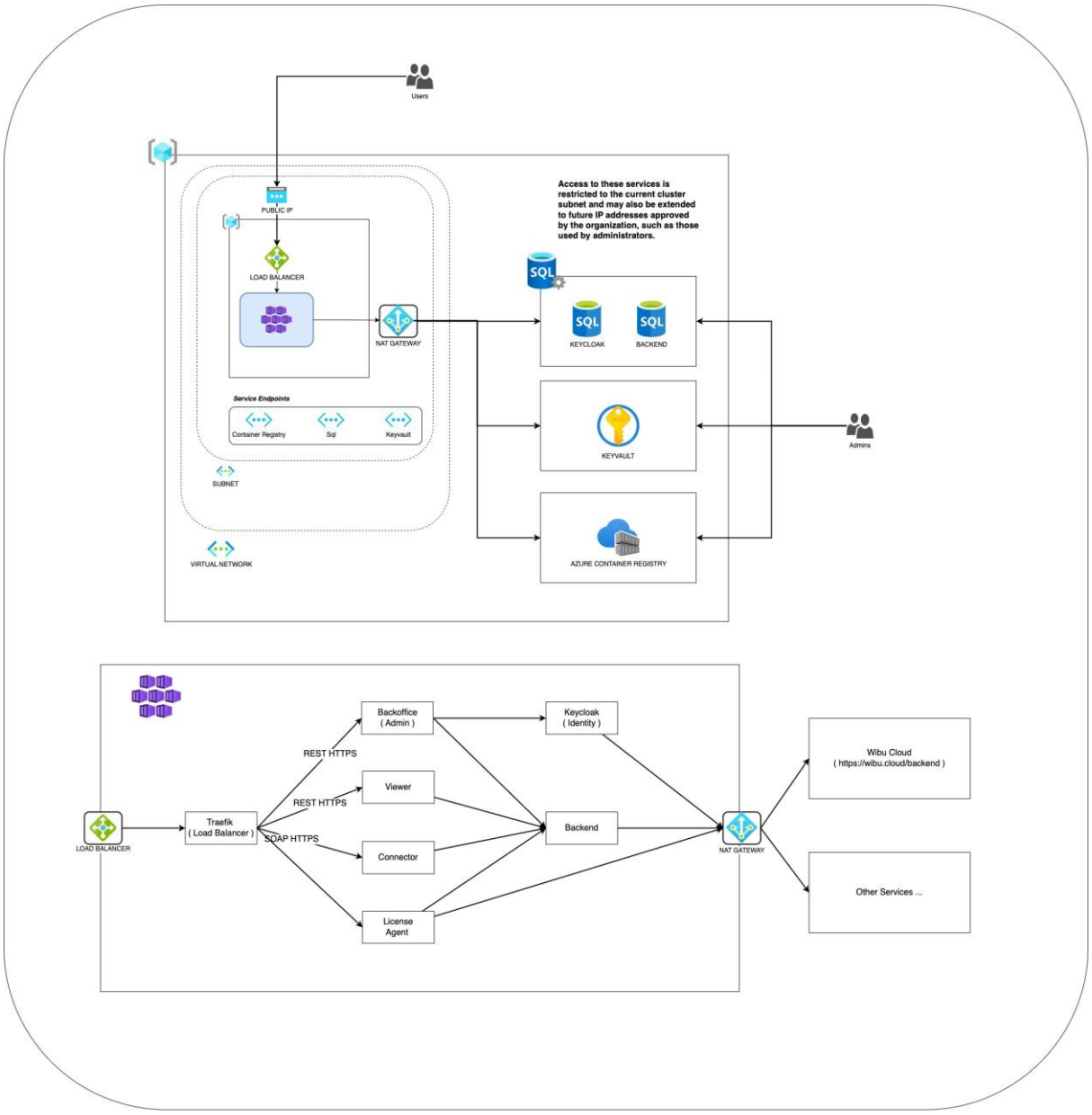
Table of Contents

Contents

Deploying DTI with Neoception® 3
PREREQUISITES 4
STEP-BY-STEP TUTORIAL 4
Step 1: Create a Resource Group 4
Step 2: Set Up Virtual Network and Subnet 4
Step 3: Allocate a Static Public IP 5
Step 4: Create a Service Principal for AKS / Keyvault 5
Step 5: Create an Azure Container Registry 6
Step 6: Create a Service Principal for ACR 7
Step 7: Deploy AKS Cluster 7
Step 8: Establish MSSQL Server and Database 8
Step 9: Create a Azure KeyVault 9
APPLICATION DEPLOYMENT USING ZARF 11
Step 1: Configure Admin Role for Kubernetes Cluster 11
Step 2: Get Kubeconfig 11
Step 3: Initialize Zarf 11
Step 4: Create Fine-grained Seccomp Profile 12
Step 4: Log In from Neoception ACR 14
Step 5: Deploy Zarf Packages 14
Core Package 15
External Secrets Package 16
Authentication Package 16
License Package 16
Backend Package 17
Backoffice Package 17
Frontend (Viewer) Package 17
Step 6: Update Zarf Packages 18
Troubleshooting 18

Deploying DTI with Neoception®: A Step-by-Step Guide

This guide, provided by Neoception®, demonstrates how to integrate the Digital Twin Infrastructure (DTI) using Azure and Zarf resources effectively. While the infrastructure is designed for easy adaptation, it's essential to customize it to meet your specific requirements.



PREREQUISITES

Before you begin, ensure the following tools are installed on your machine:

Azure CLI - https://learn.microsoft.com/en-us/cli/azure/install-azure-cli
Zarf CLI - https://docs.zarf.dev/docs/getting-started/installing-zarf
Kubectl - https://kubernetes.io/docs/tasks/tools/

STEP-BY-STEP TUTORIAL

Please execute the steps in the same order given in this guide.

Step 1: Create a Resource Group

To check the available locations, use the following command:

```
az account list-locations -o table
```

Choose a value from the **Display Name** column. Ensure to use this location for every step of this guide.

Create a Resource Group in your preferred location:

```
az group create --name "<resource_group_name>" --location "<location>"
```

Step 2: Set Up Virtual Network and Subnet

Create a Virtual Network specifying the IP range:

```
az network vnet create --resource-group "<resource_group_name>" \
                       --name "<virtual_network_name>" \
                       --address-prefixes "10.0.0.0/16"
```

Create a public IP for the Nat Gateway:

```
az network public-ip creat e --resource-group "<resource_group_name>" \
                             --name "<pip_name>" \
                             --allocation-method "Static" \
                             --sku "Standard"
```

Create the Nat Gateway:

```
az network nat gateway create --resource-group "<resource_group_name>"
                              --name "<gateway_name>" \
                              --location "<location>" \
                              --public-ip-addresses "<pip_name>"
```

```
az network vnet subnet create --resource-group "<resource_group_name>"
                              --vnet-name "<virtual_network_name>" \
                              --name "<subnet_name>" \
                              --address-prefix "10.0.0.0/20" \
                              --service-endpoints "Microsoft.Sql" "Micr
                              --nat-gateway "<gateway_name>"
```

Note: For this example, we use 10.0.0.0/16 and 10.0.0.0/20, but use the address prefix available in your organization.

Step 3: Allocate a Static Public IP
Generate a Static Public IP with the Static allocation method:

```
az network public-ip create --resource-group "<resource_group_name>" \
                            --name "<public_ip_name>" \
                            --allocation-method "Static" \
                            --sku "Standard"
```

You should have a DNS A Record with your domain pointed to this Public Ip Address. To find out the Ip Address run the following command:

```
az network public-ip show -g "<resource_group_name>" \
                          -n "<public_ip_name>" \
                          --query ipAddress -o tsv
```

Step 4: Create a Service Principal for AKS / Keyvault
Create a Service Principal for AKS creation and to give access to Keyvault:

```
az ad sp create-for-rbac --name "<display_name>" --years 5
```

If Successful, you will get a JSON that looks like this:

```
{
  "appId": "e84c2b19-e3e4-40cb-8de8-9731db45ea39",
  "displayName": "test-sp",
  "password": "r6S8Q~xOkVkcwnllF.gOUA_5AJYaK-j6j6z4tc2n",
  "tenant": "463945db-f94d-4bdf-b4e0-b60b193ad28a"
}
```

Make sure to save the appId and password values because they will be needed for future steps.

Assign a "Network Contributor" Role to the Service Principal scoped to the resource group

```
az role assignment create   --assignee "<app_id>" \
                            --role "Network Contributor" \
                            --scope <resource_group_id>
```

To get the current subscription ID, please use the following command:

```
az account show --query "id" --output tsv
```

Step 5: Create an Azure Container Registry
Create an ACR instance. The registry name must be unique within Azure and contain 5-50 lowercase alphanumeric characters:

```
az acr create --resource-group "<resource_group_name>" \
              --name "<acr_name>" \
              --location "<location>" \
              --admin-enabled false \
              --sku "Basic"
```

All access to the ACR should be denied by default

```
az acr update --name "<acr_name>" \
              --resource-group "<resource_group_name>" \
              --default-action Deny
```

Allow the cluster subnet to communicate with the ACR server

```
az acr network-rule add --name "<acr_name>" \
                        --resource-group "<resource_group_name>" \
                        --subnet "<subnet_id>"
```

Add your IP also in order to administrate the container registry

```
az acr network-rule add --name "<acr_name>" \
                        --resource-group "<resource_group_name>" \
                        --ip-address "<your_public_ip>" # or IP range
```

Step 6: Create a Service Principal for ACR
Create a Service Principal:

```
az ad sp create-for-rbac --name "<display_name>" --years 5
```

This command creates a service principal without assigning a specific role at creation. It outputs information including the *appId, secret* which you'll use in subsequent steps. Ensure to save this information.

First, find the resource ID of your ACR using:

```
az acr show --name "<acr_name>" \
            --resource-group "<resource_group_name>" \
            --query "id" --output tsv
```

Then, for each role (AcrPull and AcrPush), assign the role to the service principal:

```
az role assignment create --assignee "<appId>" \
                          --scope "<acr_resource_id>" \
                          --role "AcrPull"

az role assignment create --assignee "<appId>" \
                          --scope "<acr_resource_id>" \
                          --role "AcrPush"
```

Step 7: Deploy AKS Cluster
Launch the Kubernetes Cluster.

```
az aks create --resource-group "<resource_group_name>" \
              --name "<cluster_name>" \
              --location "<location>" \
              --node-count 1 \
              --kubernetes-version "1.29.0" \
              --node-vm-size "Standard_B2ms" \
              --max-pods 250 \
              --dns-name-prefix "<cluster_name>" \
              --generate-ssh-keys \
              --network-plugin "azure" \
              --network-policy "azure" \
              --service-principal "<appId>" \
              --client-secret "<password>" \
              --api-server-authorized-ip-ranges "<api_server_authorize
              --vnet-subnet-id "<vnet_subnet_id>" \
              --outbound-type userAssignedNATGateway \
              --ssh-key-value "<rsa_public_ssh_key>" \
              --load-balancer-sku standard
```

If you want high-availability in your system please make sure that *--node-count* is equal to 3. For *--api-server-authorized-ip-ranges*, please put the public IP addresses of the machines that will

administrate this Kubernetes Cluster separated by commas. Finally add a public rsa ssh key in order to access the node, it will be necessary to create some security profiles

To retrieve the subnet ID required for --vnet-subnet-id, use:

```
az network vnet subnet show --resource-group "<resource_group_name>" \
                            --vnet-name "<vnet_name>" \
                            --name "<subnet_name>" \
                            --query "id" --output tsv
```

Step 8: Establish MSSQL Server and Database
Set up the SQL Server with an admin username and password:

```
az sql server create --location "<location>" \
                     --name "<server_name>" \
                     --resource-group "<resource_group>" \
                     --admin-user "<admin_user>" \
                     --admin-password "<admin_password>"
```

Configure Firewall Rules to allow connections from your personal computers:

```
az sql server firewall-rule create --name "<firewall_rule_name>" \
                                   --resource-group "<resource_group_n
                                   --server "<sql_server_name>" \
                                   --end-ip-address "<ip_address>" \
                                   --start-ip-address "<ip_address>"
```

For this, you should add a firewall rule with your Public IP address to manage your MSSQL databases (you will need to create a user for each database, so don't skip this step).

Create the Databases tailored for your backend and your Keycloak instances:

```
az sql db create --name "<database_name>" \
                 --resource-group "<resource_group_name>" \
                 --server "<sql_server_name>" \
                 --read-scale "Disabled"
```

Note: Replace <*database_name*> with "backend" and "keycloak" for each respective command.

After this, you should have 3 databases inside your SQL server:

- master
- backend
- keycloak

To get the connection string for each database, run the following command:

```
az sql db show-connection-string --client "ado.net" \
                                 --name "<database_name>" \
                                 --server "<sql_server_name>"
```

Fill in the missing parameters (*<database_name>*, *<username>*, and *<password>*).

First, log into your master database, where you will create the database logins for your users:

```
CREATE LOGIN backendlogin WITH password='<random_password>'; CREATE
LOGIN keycloaklogin WITH password='<random_password>';
```

After this, log into each of your databases and create the user with the db_owner role:

```
CREATE USER <user> FROM LOGIN <login>;
EXEC sp_addrolemember 'db_owner', '<user>';
```

Create an Azure SQL Server Virtual Network (VNet) rule to secure your Azure SQL database by controlling inbound traffic from selected subnets within a virtual network. This security measure is crucial for ensuring that only authorized network traffic can access your database services:

```
    az sql server vnet-rule create \
        --name "sql-vnet-rule" \
    --server "<your_sql_server_name>" \
    --resource-group "<resource_group_name>" \
    --subnet "<your_subnet_id>"
```

Step 9: Create a Azure KeyVault
Create an Azure KeyVault using the following command:

```
az keyvault create --name "aas-keyvault-test" \
                   --resource-group "aas" \
                   --location "West Europe" \
                   --network-acls-vnets "aas-vnet/aas-cillium-subnet"
                   --bypass "None"
```

The output of this command shows properties of the newly created key vault. Take note of the Vault Uri property. Next create an access policy for the previous created Service Principal with the following permissions:

```
az keyvault set-policy --name <keyvault_name> \
                       --object-id <sp_object_id> \
                       --secret-permissions get list set delete backup
```

To get your Service Principal Object Id, run the following command:

```
az ad sp show --id <sp_app_id> --query id --output tsv
```

To Finalize our infrastructure Setup please define the following secrets:

```
# Authentication App Secrets

# The admin password for your Keycloak Instance
az keyvault secret set --vault-name <keyvault_name> \
                       --name "KEYCLOAK-ADMIN-PASSWORD" \
                       --value <value>
# Your JDBC connection string for Keycloak - e.g jdbc:sqlserver://<seraz
keyvault secret set --vault-name <keyvault_name> \
                       --name "KC-DB-URL" \
                       --value <value>
# The username of your keycloak DB - if you followed the guide it shouaz
keyvault secret set --vault-name <keyvault_name> \
                       --name "KC-DB-USERNAME" \
                       --value <value># The
password of your keycloak DB
az keyvault secret set --vault-name <keyvault_name> \
                       --name "KC-DB-PASSWORD" \
                       --value <value>
```

```
# The URL of your backoffice instance - e.g admin.your.domain.comaz
keyvault secret set --vault-name <keyvault_name> \
                       --name "KC-AUTH-URL" \
                       --value <value>

# Backend App Secrets

# The connection string of your backend database
az keyvault secret set --vault-name <keyvault_name> \
                       --name "CONN-STRING" \
                       --value <value>
# The Auth URL used by your instance e.g - https://admin.your.domain.caz
keyvault secret set --vault-name <keyvault_name> \
                       --name "BACKEND-AUTH-URL" \
                       --value <value>
```

APPLICATION DEPLOYMENT USING ZARF

Step 1: Configure Admin Role for Kubernetes Cluster

There are two Azure roles you can apply to a Microsoft Entra user or group:

- Azure Kubernetes Service Cluster Admin Role
- Azure Kubernetes Service Cluster User Role

For this deployment, you will need to give Admin permissions to your user to deploy Zarf and its packages:

```
# Get the resource ID of your AKS cluster
AKS_CLUSTER=$(az aks show --resource-group "<resource_group_name>" \
                --name "<aks_name>" \
                --query "id" -o tsv)

# Get the account credentials for the logged-in user ACCOUNT_UPN=$(az
account show --query "user.name" -o tsv) ACCOUNT_ID=$(az ad user show --id
"$ACCOUNT_UPN" --query "id" -o tsv)

# Assign the 'Cluster Admin' role to the user
az role assignment create --assignee "$ACCOUNT_ID" \
                          --scope "$AKS_CLUSTER" \
                          --role "Azure Kubernetes Service Cluster Adm
```

Step 2: Get Kubeconfig

Get the kubeconfig file to access your Kubernetes Cluster:

```
az aks get-credentials --resource-group "<resource_group_name>" \
                       --name "<aks_cluster_name>" \
                       --admin
```

Verify that the admin configuration information has been applied using:

```
kubectl config view
```

Step 3: Initialize Zarf

To initialize Zarf, set these variables to use the ACR as the main registry:

```
export ACR_URL="<acr_login_server>"
export ACR_USERNAME="<acr_sp_object_id>" # ACR Service Principal Object
export ACR_PASSWORD="<acr_sp_password>" # ACR Service Principal Password
```

To get the ACR login server, run this command

```
az acr show -n "<acr_name>" \
            -g "<resource_group_name>" \
            --query "loginServer"
```

After the variables are established, run this command to initialize Zarf:

```
         zarf init --registry-push-password="${ACR_PASSWORD}" \
                   --registry-push-username="${ACR_USERNAME}" \
          --registry-url="${ACR_URL}"
```

You will be prompted with these questions:

- Do you want to download the Zarf Init package? - Yes
- Do you want to deploy the Zarf Init package? - Yes
- Do you want to deploy the logging Component? – No
- Do you want to deploy the Gitea server Component? - No

Step 4: Create Fine-grained Seccomp Profile
If you have more than one node, please make sure to replicate this process to the other ones. To list your nodes use the following command:

```
kubectl get nodes -o wide
```

Output should be something like this

```
NAME                                STATUS   ROLES   AGE    VERSION   I
aks-nodepool1-37663765-vmss000000   Ready    agent   166m   v1.25.6   1
aks-nodepool1-37663765-vmss000001   Ready    agent   166m   v1.25.6   1
```

Use the kubectl debug command to start a privileged container on your node and connect to it.

```
kubectl debug node/<your-node-name> -it \
        --image=mcr.microsoft.com/cbl-mariner/busybox:2.0
```

Sample Output

```
Creating debugging pod node-debugger-aks-nodepool1-37663765-vmss000000with
container debugger on node aks-nodepool1-37663765-vmss000000.
If you don`t see a command prompt, try pressing enter.
/#
```

After this, run the following command to interact with your node

```
/# chroot /host
```

Navigate to /var/lib/kubelet/seccomp and create following fine-grained.json file in this directory:

```
{
    "defaultAction": "SCMP_ACT_ERRNO",
    "architectures": [
        "SCMP_ARCH_X86_64",
        "SCMP_ARCH_X86",
        "SCMP_ARCH_X32"
    ],
    "syscalls": [
        {
            "names": [
                "accept4",
                "epoll_wait",
                "pselect6", "futex",
                "madvise", "epoll_ctl",
                "getsockname",
                "setsockopt", "vfork",
                "mmap",
                "read",
                "write",
                "close", "arch_prctl",
                "sched_getaffinity","munmap",
                "brk", "rt_sigaction",
                "rt_sigprocmask",
                "sigaltstack", "gettid",
                "clone",
                "bind",
                "socket",
                "openat", "readlinkat",
                "exit_group",
                "epoll_create1","listen",
                "rt_sigreturn",
                "sched_yield",
                "clock_gettime",
```

```
                "connect", "dup2",
                "epoll_pwait","execve",
                "exit",
                "fcntl",
                "getpid",
                "getuid",
                "ioctl", "mprotect",
                "nanosleep","open",
                "poll", "recvfrom",
                "sendto",
                "set_tid_address",
                "setitimer", "writev",
                "fstatfs", "getdents64",
                "pipe2", "getrlimit"
            ],
            "action": "SCMP_ACT_ALLOW"
        }
    ]
}
```

If the folder seccomp doesn't exist, make sure to create it

After this exit your container by typing exit two times , one for exiting the node and other exiting the container

Step 4: Log In from Neoception ACR
To access the required Zarf packages, you must log in to the Neoception Container Registry using credentials provided to you. These credentials provide read-only access.

```
echo "${NEO_ACR_PASSWORD}" | zarf tools registry login "dtineoceptiona
                              --username "${NEO_ACR_USERNAME}" --passw
```

Step 5: Deploy Zarf Packages
Install the Zarf packages in the following order for optimal setup:

1. Cor
2. External Secrets
3. Authentication
4. Connector
5. License
6. Backend

7. Backoffice
8. Frontend

Please make sure you read the information for each component, if there are any additional steps, do those first before deploying the package.

To deploy the package run the following command:

```
zarf package deploy oci://dtineoceptionacr.azurecr.io/<package-name>
```

For some packages Zarf will prompt for variable values.

Core Package
Description: This combined package deploys Traefik ingress controller, Cert-Manager, Reloader and Opa Gatekeeper using Helm charts to manage external access to services and automate the management of SSL/TLS certificates in a Kubernetes cluster.

Variables:

- **LB_RESOURCE_GROUP** (Traefik-specific): The name of the resource group where the Kubernetes cluster resides. Essential for configuring Traefik with specific cloud resources.
- **LB_PUBLIC_IP** (Traefik-specific): The Public IP address that Traefik should utilize for handling inbound traffic.
- **EMAIL_NOTIFY** (Cert-Manager-specific): An email address to receive notifications about certificate expiration dates, enhancing the monitoring and maintenance of SSL/TLS certificates.
- **ALLOWED_REGISTRY** (OPA Gatekeeper specific): The ACR registry where all images should come from (in this case, your ACR)

Components:

- Traefik - https://traefik.io/traefik/
- Cert-Manager - https://cert-manager.io/
    1. ClusterIssuer - Kubernetes resource that represent certificate authorities (CAs) that are able to generate signed certificates by honoring certificate signing requests
- Reloader - https://github.com/stakater/Reloader
- Opa Gatekeeper
    1. Templates1.1 https://github.com/openpolicyagent/gatekeeperlibrary/blob/master/library/general/block-nodeport-services/template.yaml

1.2
https://github.com/openpolicyagent/gatekeeperlibrary/blob/master/library/general/allowedrepos/template.yaml

1.3

https://github.com/openpolicyagent/gatekeeperlibrary/blob/master/library/general/disallowedtags/template.yaml

---

External Secrets Package
Description: Deploy the External Secrets Operator Helm Chart for secure secrets management in Kubernetes clusters using Azure Key Vault.

Variables:

- **AZURE_SP_CLIENT_ID:** Application ID of the service principal.
- **AZURE_SP_SECRET:** Secret password of the service principal.
- **AZURE_VAULT_URI:** URI of the Azure Key Vault.
- **AZURE_TENANT_ID:** Azure Subscription Tenant ID.

Components:

- External Secrets Operator - https://external-secrets.io/
  1. Cluster Secret Store
  2. Backend and Keycloak External Secrets

Authentication Package
Description: Set up an authentication system for DTI using Keycloak.

Components:

- Keycloak - https://www.keycloak.org/
- Auth Image - Neoception Docker image containing Realm and Theme configurations for Keycloak

---

License Package
Description: Deploy the DTI license management system to a Kubernetes cluster.

Components:

- License Image - Custom Docker Image containing Wibu-Software CodeMeter License Central that lets you create, manage, and distribute licenses

Setup Pre-Installation
It will be given to you a credential file, where you will need to create a configmap with the content of it. For it just create a file named kustomization.yaml and put this yaml configuration inside of it.

```
configMapGenerator:
  - name: credconfig
    namespace: dti files:
      - ./<file_name>.wbc
generatorOptions:
  disableNameSuffixHash: true
```

Make sure that kustomization.yaml and the credential file are in the same directory. Run the following command to create the configmap:

```
kubectl create -k . -o yaml --dry-run=client > configmap.yaml
```

After the creation , deploy it to the cluster:

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

Backend Package
Description: Deploy the backend component of the DTI environment.

Components:

- Backend - Neoception Docker image containing the backend application for DTI

Backoffice Package
Description: Deploy the Backoffice application for DTI.

Variables:

- BACKOFFICE_URL: URL where the Backoffice instance will be accessible, e.g., admin.your.domain.com.

Components:

- Backoffice - Neoception Docker image containing the backoffice application for DTI

Frontend (Viewer) Package
Description: Add the Web UI viewer application to the DTI cluster.

Variables:

- VIEWER_URL: URL where the viewer instance will be accessible, e.g., viewer.your.domain.com.

Components:

- Viewer - Neoception Docker image containing the viewer application for DTI

Step 6: Update Zarf Packages
To update your applications, simply deploy the new package version (which will be provided to you). Zarf handles updates automatically.

Please ensure that all command parameters are correctly replaced with your specific values before execution.

Troubleshooting
If any Zarf related step fails to complete you may run it with l=trace at the end and send us the output for support on what is going wrong.