# D:\__jk1510_workspace\Projects\Temenos-ABN\temenos.png

Stack 9 Azure

CI/CD Pipeline approach v1.3

20-December-2020

Contents

[1. Introduction 3](#_Toc59371162)

[1.1. Document Purpose 3](#_Toc59371163)

[1.2. Definitions 3](#_Toc59371164)

[2. Current System State 3](#_Toc59371165)

[2.1. System Components 3](#_Toc59371166)

[2.2. Deployment Methodology 4](#_Toc59371167)

[2.2.1. Branching Strategy 4](#_Toc59371168)

[2.2.2. Build Process 5](#_Toc59371169)

[2.2.3. Quality Assurance 5](#_Toc59371170)

[2.2.4. Release Process 5](#_Toc59371171)

[2.2.5. L1 & L3 Pipelines 5](#_Toc59371172)

[3. Proposed System 5](#_Toc59371173)

[3.1. System components 6](#_Toc59371174)

[3.2. System Development 7](#_Toc59371175)

[3.3. Proposed Deployment Methodology 8](#_Toc59371176)

[3.4. Branching Strategy 8](#_Toc59371177)

[3.5. Build Pipeline 9](#_Toc59371178)

[3.6. Release Pipeline 9](#_Toc59371179)

[3.7. Azure DevOps Dashboard 10](#_Toc59371180)

[3.8. Security Centre 11](#_Toc59371181)

[3.9. Container Monitoring 12](#_Toc59371182)

[4. T24Updates & L3 Deployment Approach 13](#_Toc59371183)

[4.1. Approach-I 13](#_Toc59371184)

[4.1.1. Process Flow 15](#_Toc59371185)

[4.1.2. T24/Packages Pipeline Stages 17](#_Toc59371186)

[4.2. Approach-II 17](#_Toc59371187)

[4.2.1. Process Flow 18](#_Toc59371188)

[4.2.2. T24/Packages Pipeline Stages 20](#_Toc59371189)

[5. Challenges 20](#_Toc59371190)

[6. Assumptions 21](#_Toc59371191)

[7. Considerations 21](#_Toc59371192)

[7.1. Availability 21](#_Toc59371193)

[7.2. Persistent Volume 21](#_Toc59371194)

[7.3. Scalability 21](#_Toc59371195)

[7.4. Security 21](#_Toc59371196)

[7.5. Resiliency 21](#_Toc59371197)

# I**ntroduction**

## Document Purpose

The purpose of this document is to describe the high-level solution for implementing continuous integration and continuous deployment of T24 Transact in AKS cluster.

## Definitions

The following acronyms and definitions will be used in this document

|  |  |
| --- | --- |
| Acronym | Description |
| Pipeline | Azure DevOps pipelines (Build & Release) |
| Artefacts | T24 binary components |
| Image | A Container image |
| Container Registry | An azure service for storing container images |
| Source Code Repository | A Bitbucket repository |
| Components Repository | A Nexus repository |
| Package Folders | A pre-image kit folders in the Nexus repository |
| Pods | A container deployable units |
| Cluster | An Azure AKS cluster |

# Current System State

Currently, the team is following in-house DevOps practice called One Release Strategy to simplify the release process and speed up software releases. The main objective is emerge shift left approach and maintains one release branch at a time. With the ORS process, the team benefits following:

* Sprint delivery on time
* Maintains latest and greatest version of the code
* Identical deployment in all environments
* Focused release at a time
* Identify the bugs in early stage

## System Components

T24 technology components and L1/L3 packages are deployed using the automated pipelines configured using Jenkins & XL release. However, there are quite inevitable manual processes involved.

**Bitbucket**

Bitbucket is the primary source code management system with repositories for application configuration. There are component specific repositories used within the system. The following shows the repository structure.

|  |  |
| --- | --- |
| Pipeline config | This folder contains the XL release pipeline configuration and XL release templates |
| Env dictionary files | This folder contains the environment specific config files and dictionary files |
| T24 Application folder | The application .ear & .war files are stored here |
| T24 Application config files | The configuration folder contains the environment specific config files and JVM options |
| T24 Application DAR | This folder contains the DAR pom file used in the deployment orchestration |

**Jenkins**

Jenkins used as the orchestrators for building the T24 components such as Browser, TAFJEE, Axis2, Etc. Also, the pipelines are integrated with XL release to create new releases from the templates which are stored in the repository.

**Nexus Repository**

Nexus is used as a primary centralized storage repository for T24 binaries and configuration files. The same has been integrated with the XL release pipeline for seamless artefacts management.

**XL Release**

It helps to plan, track and execute release plans in the environment. The new release plans are created as ad-hoc basis from Jenkins build.

## Deployment Methodology

The components/products are deployed in various environments using linear strategy. The same artefact is promoted between all stages when the previous deployment is successful. As per our understanding, components deployments traverse through following environments. The stages might differ from release type’s \* i.e. normal, hotfix

* POC/Dev
* ST
* RT
* ET
* PROD

### Branching Strategy

The branch creation is pretty straight forward at the moment. Master branch is a stable one which has the latest and greatest code of production environment. The develop branch is an intermediate one which gets created from master. When the development is over, release branch will be created from this branch. Hotfix branch is created out of master when patch release needs to be pushed. Develop & release branches are owned by the delivery teams whereas the master & hotfix branches are owned by upgrade team.

### Build Process

The T24 components are less likely to be compiled since the product binary doesn’t allow changes whereas the local development packages are compiled locally and the build artefacts are pushed to centralized nexus repository.

### Quality Assurance

There are plenty of test cases are executed in each stage. The testing team starts writing the test cases in parallel with development activity. Therefore, the test steps are aligned with the requirement most of the time and test cases are approved internally before executing in any environment.

### Release Process

The typical component deployment flow would be:

A developer creates new branch in the components specific repository. Ideally, the new branch will be created from master; otherwise, the latest release branch. Once the new branch is created, the developer will clone the repository into their local development environment and start pushing custom changes. When the development is complete, the Pull Request will be created from current working branch to destination branch (e.g. master). Eventually, all the members who are working in CI/CD will be added as a reviewer and based on no conflicts; the PR will be merged to the destination branch.

Each components has own Jenkins job, the builds are triggered manually. The Jenkins job reads the config files from the repository and creates a new release in the XL Release. The corresponding release pipeline URL will be populated in the console output for monitoring purposes. Authorised users can navigate through the XL release link and monitor the deployment status. The XL release pipeline steps will be executed in sequence. The first step is parses the config options i.e. loading environment variables. And, update XL Deploy which is doing some environment preparation steps. Finally, the main deployment will be triggered in respective environment based on the pipeline\_config.json.

### L1 & L3 Pipelines

The T24 updates and L3 DS package deployment also got deployed using an automated pipelines. In an ideal scenario, the t24 update packages are stored in a specific Bitbucket repository and during the run time, it gets deployed into target environment and also it is uploaded to Nexus repository. The design studio packages are developed locally and published to Nexus directly and then through the dedicated release pipeline, it will be deployed into the target environment.

# Proposed System

Enabling continuous integration and deployment for containers is not straight forward as standalone environments since containers hold only immutable data and restrictions on the access layers. To achieve the automated pipelines, we need to choose the correct deployment strategy based on the application eco-system. Especially, for the T24 Transact, the components are tightly coupled so the required components must be packaged in the pre-image kit upfront and the container images need to be built out of it. We are going to create the container images with deployable components rather from an application source code. We need to split the container image build and deploy process; therefore, we can publish the newer images into the remote container registry anytime without worrying about Instant deployments. Eventually, the automated deployment must be configured for the DEV environments whereas other environments deployment requires automated/manual approval from the respective environment owners.

In our solution approach, we have covered the entire Transact deployment in a single cluster, and T24 updates on a container methodology as well. We still are going to follow the ORS process with underlying changes on the automation tool stack.

## System components

In the new deployment architecture, we will use end to end Microsoft services to achieve the automated build and release process.

**Azure Repos**

The components configuration files and deployment scripts will be stored in the azure repositories. We can maintain the same folder structure as in the existing Bitbucket repository. However, considering deployment methods and number of components, we can refactor the structure.

**Azure Pipelines**

Azure DevOps pipelines will be used for building container images and publishing that to ACR registry. This pipeline will be integrated with Nexus repository for downloading getting the required pre-image kit files. The scripted azure pipeline will be stored in azure repository.

**Azure Releases**

An automated Azure DevOps pipeline, as the name denotes, this will be used for release/deploying artefacts. The custom deployment tasks can be configured. Moreover, the manual approval process will be introduced in between environment deployments.

**Azure container registry**

The T24 transact container images will be pushed and centrally managed in Azure container registry. We will follow the existing naming standards for tagging new images. The ACR should be configured with Premium tier for enabling Replication feature, it is highly recommended to replicate the production images into paired region.

**Helm**

Helm charts are the deployment orchestrator for Kubernetes. The pod definition files will be in the form of helm charts and this will be updated now and then with the latest change set.

**Nexus Repository**

The technology components binary will be stored in Nexus repository. To build the new container images, those binaries will be pulled during runtime and packages with pre-image kit.

**Azure Build Agents**

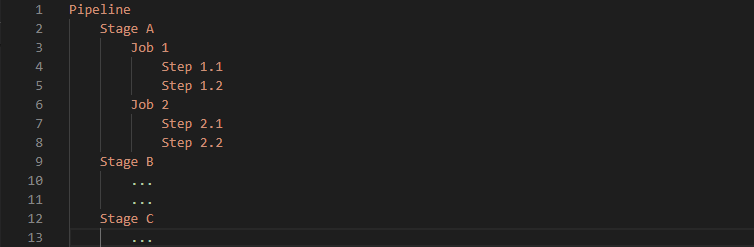
It is important to configure the agent pools to our pipeline jobs. Microsoft hosted agent is managed instance in Azure DevOps organization. There will be one free agent with 1800 pipeline minutes per month. We can also configure self-hosted agent with either standalone VM / Azure Scale Sets.

**Azure Monitor**

By default AKS monitoring can be configured by deploying some add-ons. The OMS agent pods and Log analytics workspace will be created when enabling this feature. This will be useful to capture cluster access logs and system logs.

## System Development

In the new approach, there is quite a bit of automation script development efforts involved. Starting from helm charts, custom scripts to pipeline configurations. Helm charts will define the kubernetes deployment structure and helps to maintain the versioning of our scripts. Moreover, it facilitates the rollback feature out of the box which can be configured in the release pipeline. The Azure pipeline will have more stages to describe the CI/CD process for deployments of technology components, T24 updates and L3 DS packages. It is recommended to define the pipeline in a YAML configuration. The pipeline structure will look like;



Pipeline script, helm chart and other automation scripts will be stored in an Azure repository “deployment\_repo”. The scripts must be generic, in a way that environment specific variables will be configured in the Azure Devops variables groups and referenced in the pipeline.

*Here we are referring to Transact and T24 updates, L3 deployment scripts on existing azure infrastructure. Assuming azure infrastructure provisioning will be handled separately.*

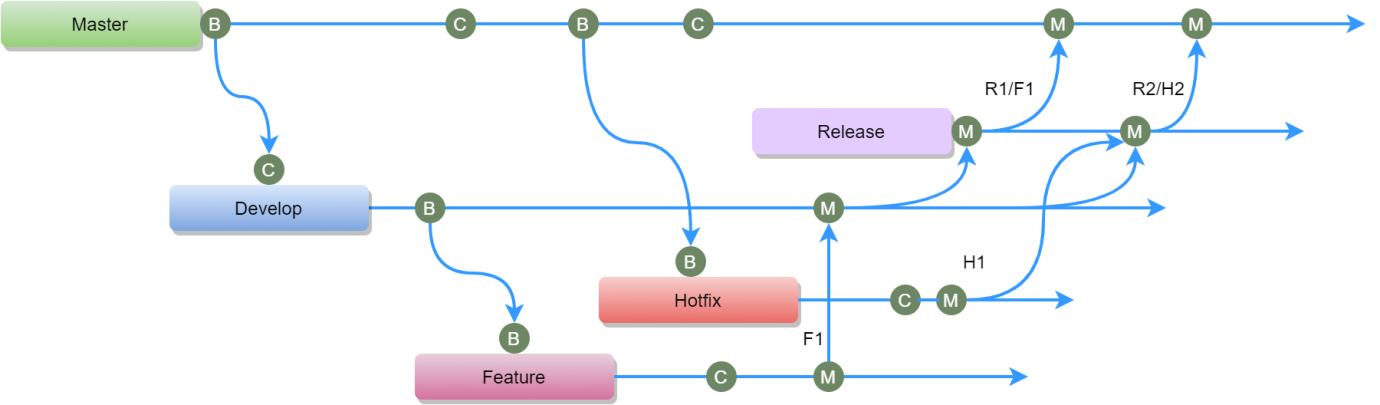
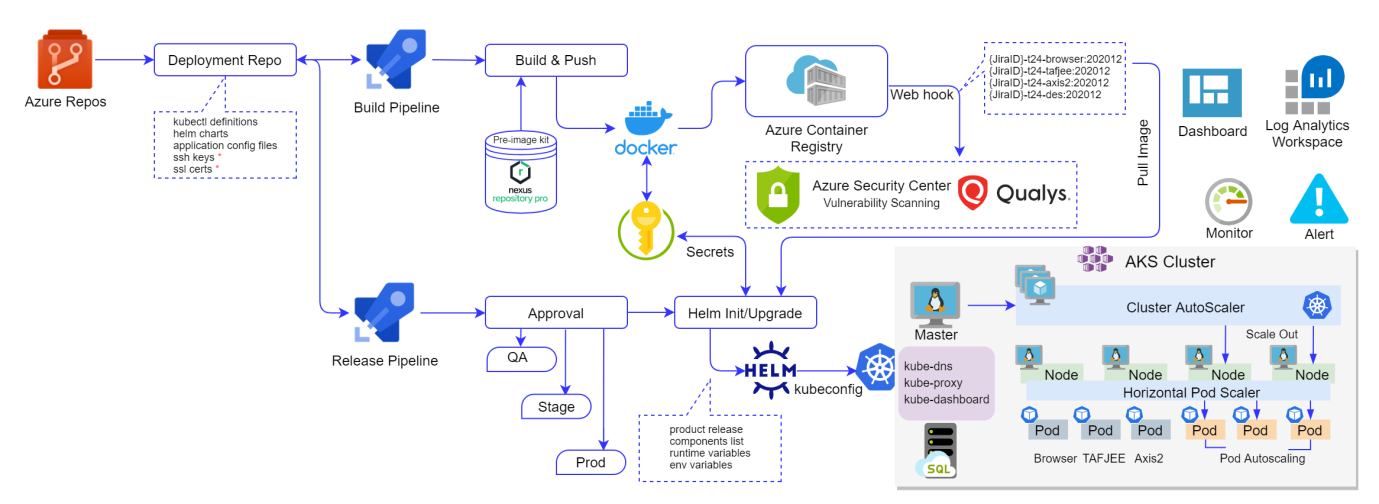
## Proposed Deployment Methodology

The below diagram depicts the T24 transact build and deployment process.

*Figure-1.1 Build & Release*

## Branching Strategy

The first and foremost step in automating the application release is an effective repository structure and branching strategy. The recommended branching technique for agile development as follows.



*Figure-1.2 Branching Strategy*

Every developer & tester will commit their code in the Feature branch and based on the peer review it will be merged into the development branch. After testing is done, the develop branch will be merged to a release branch. We can retain the release branch for every sprint just in case if want to rollback/revert the changes. Otherwise, after a retention period, this branch will be archived. From release branch it will be merged to master branch, this merge will happen after the successful release. Thus, the master branch will always have the latest and greatest code. The hot-fix branch will be created from the master or last known successful release tag and follow the same process as a develop branch.

*The above is planned adhere to the existing branching policy. If there are any deviations then we can refactor the same later point of time.*

We will create a new repository for versioning Helm charts, and any other automation scripts. This repository access will be allowed only for CI/CD team.

## Build Pipeline

As stated previously, the container images need to be built separately and then published to the centralized container registry. The necessary component needs to be downloaded and included in the pre-image kit automatically. The container images will be tagged with existing naming standards followed in ORS process. i.e. JIRA ID prefixed/suffixed with the release version. To ensure the container image security, we will enable Qualys vulnerability scanning which is an embedded feature in the ACR.

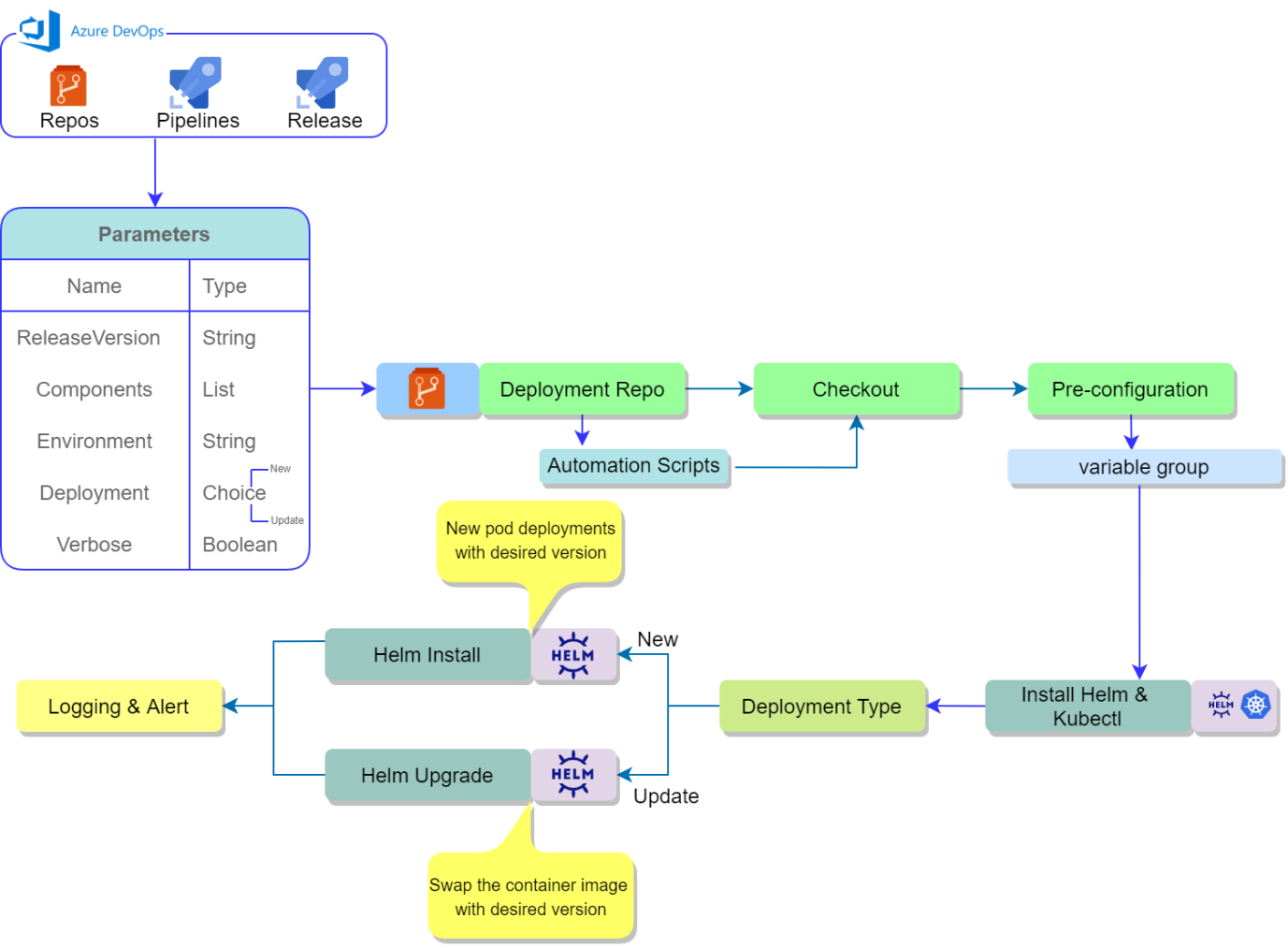
## Release Pipeline

The release pipeline is the actual deployment pipeline which will pull the desired image from the ACR and run the containers. We will introduce the manual approval process in the release pipeline to control the deployments. If there are any existing approval processes in place then we can adopt the same in our release pipeline. Otherwise, for QA/Test environments the QA lead needs to approve, Staging environments deployment will be approved by Delivery/Operations team. Production deployment will be approved by CAB board.

*Helm chart will play a key role in this pipeline. Helm chart will define the deployment structure for T24 Transact.*

This must be a generic pipeline to handle multi-environment deployments (DEV, TEST, STAGE and PROD). The below diagram illustrates the detailed view of the release pipeline.

The pipeline will be parameterized with target product version, environment type, components list (this will be required to control the single / multiple components deployment) and deployment type. The core tasks are Install the helm & kubectl packages, helm install and helm upgrade. We can also include helm rollback which is available in the recent stable releases of Helm. But this may only update the cluster not any database changes. The pre-requisites packages installation can be ignored if we use self-hosted agents so that the deployment time can be saved.



*Figure -1.3 Release Pipeline*

Ideally, the release pipeline will have required parameters such as,

Release version 🡪 Product release version e.g. R21.01. The same will be used for tagging the image

Target environment 🡪 Target AKS cluster name

Components list 🡪 Comma separated components list

Deployment 🡪 New will deploy all the components. An update will re-create only specific pod

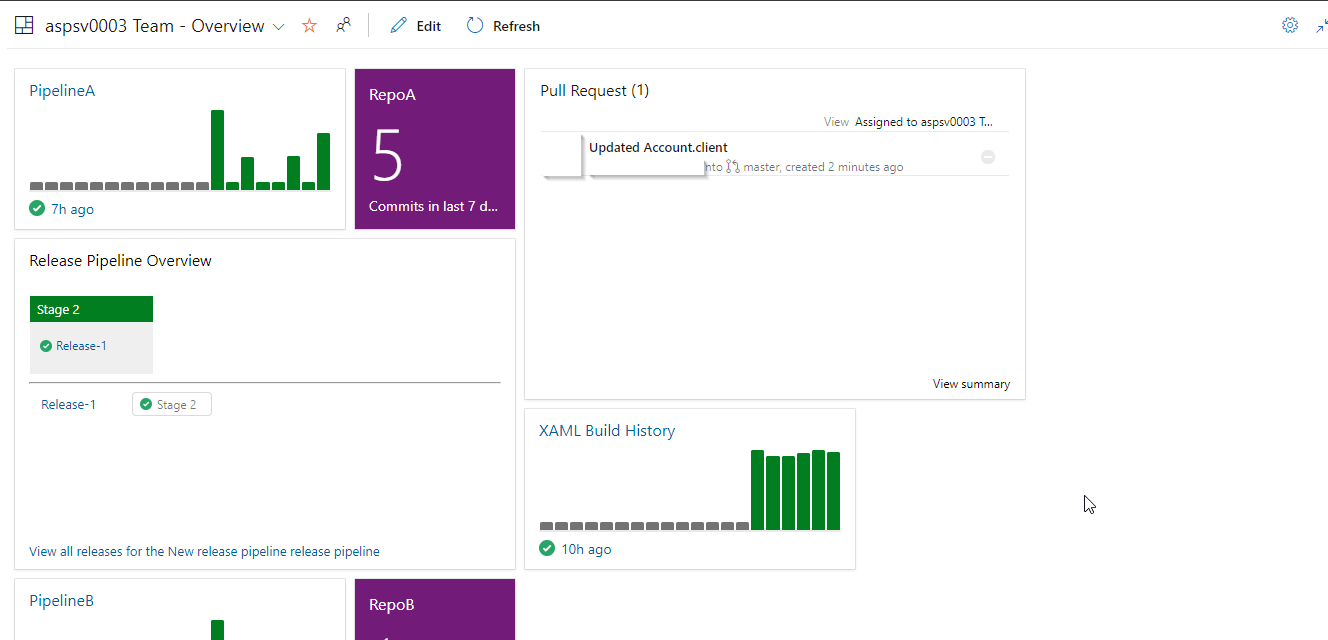
Verbose 🡪Enable detailed logging in the pipeline execution

## Azure DevOps Dashboard

Custom metrics will be useful to gain visibility into the team progress, pipeline status, etc. Azure DevOps trends for the repository, scrum board and build pipeline needs to be created. There are few default widgets available, however; we can create our dashboard with custom queries.

* Deployment Status
* Release Pipeline Overview
* Pull Requests
* Build a history chart

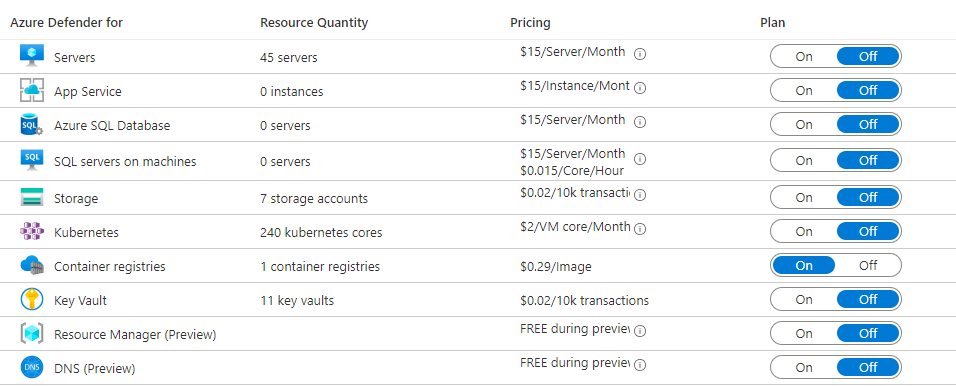
An example Dashboard which shows Pipeline status, Release Pipeline overview, Repository commits and Pull requests. There are plenty of other extensions available from Azure DevOps market place just in case if we want to configure more metrics (<https://marketplace.visualstudio.com/>)



*Figure -1.4 Sample Azure DevOps Dashboard*

## Security Centre

The azure defender service must be enabled in order to use the container vulnerability scanning feature. Even through it is an optional; if we have not integrated any other security solution to identify the potential vulnerabilities in the container images then we can make use of this in-built service. The list of service covered by defender and its pricing is shown below. We can toggle the plan on/off based on demand.

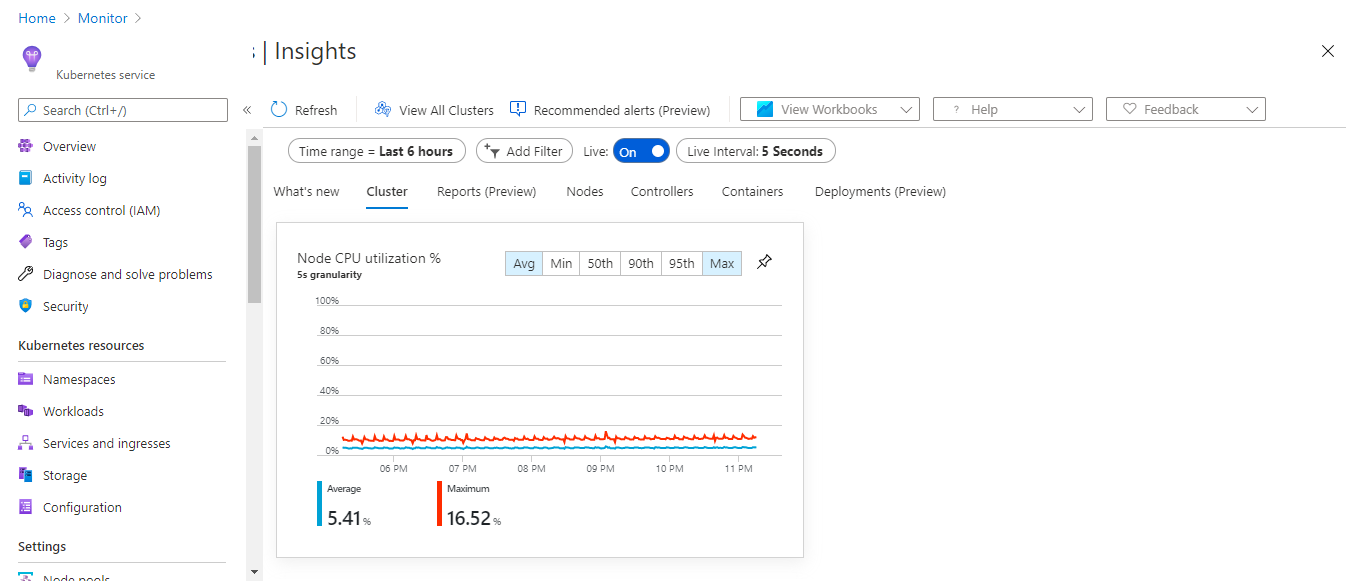


*Figure -1.5 Security Centre*

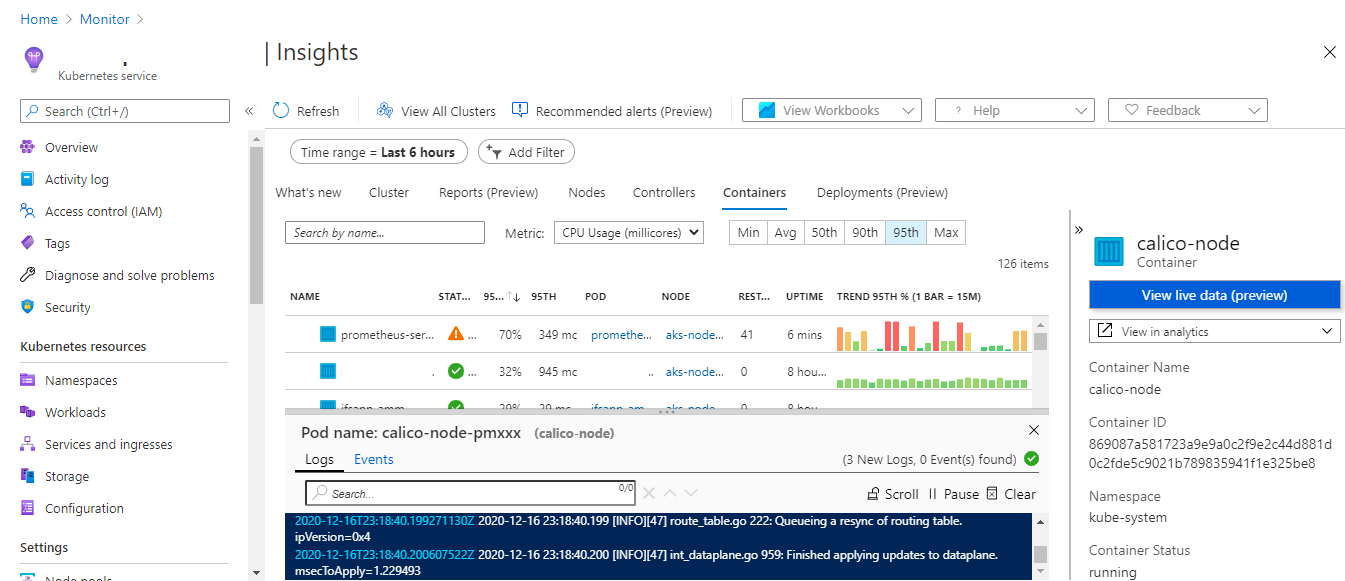
## Container Monitoring

Azure monitor has recently embedded with new features for container monitoring. The AKS cluster level monitoring must be enabled to consume these features. The comprehensive solution helps to monitor

* Worker node Count, CPU, Memory utilization
* AKS containers running on the worker node and their CPU/Memory usage
* Deployments, Pods, Replica Sets status
* View live data from the containers
* Create workbooks/dashboard with custom metrics



*Figure -1.6 Container Insights 1*



*Figure -1.7 Container Insights 2*

# T24Updates & L3 Deployment Approach

Since T24updates and L2 DS package deployments require some post configurations, we need to slightly deviate from the traditional process and adopt what is best for the container-based environments. Enabling persistent volume for the containers gives the ambiguity of running automation scripts in standard environments.

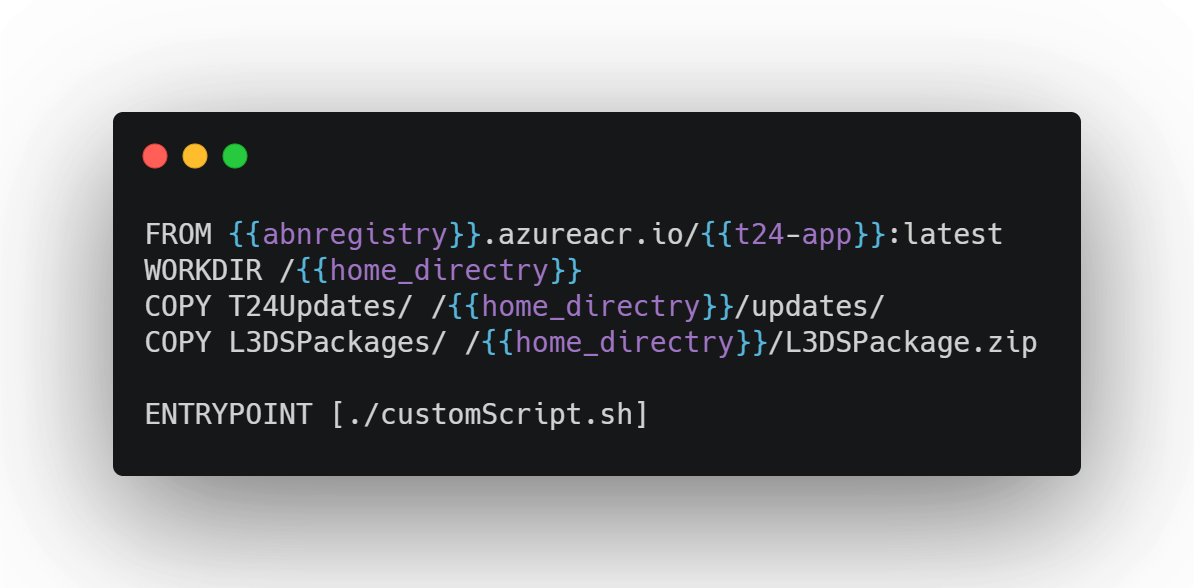
In the normal deployments, alpine/JBOSS will be used as a base image for running the containers whereas, for the t24 updates and l3 package deployments, we have to use the latest version of container images and push our custom packages on top that. This approach may be overkill with the PV volume; however, we need to make sure the container has all the previous updates and configuration to deploy the new T24 updates.

Updates & Package deployment can also be integrated with the release pipeline but considering the complex process involved in running custom scripts, it is better to have a separate pipeline for doing this job. We need to have separate folders for T24updates & L3DSPackages in the pre-image kit and the respective files will be copied to the container working directory.

There are couple of approaches to install t24 updates and DS packages through an automated pipeline.

## Approach-I

Use the application image as the base and push the updates & packages from pre-image kit to build the new container image with persistent volume. In this approach we will have only latest T24 updates / DS Packages.



*Figure-1.8 Approach I Dockerfile*

This Dockerfile will be used only for deploying T24 updates and L3 DS packages on an existing cluster. Assuming, the image in the remote registry already has the previous changes of application. The image will have the desired updates and custom script which does the post configurations. This process is explained in the following sections.

*The updates and package deployments needs to be executed on a transient/sidecar pod. Importantly, the sidecar container should be connected to the same database & PV volume must be mounted.*

The main objective of mounting the persistent volume is to preserve the updates/packages which are deployed previously. When the new updates is getting installed in the environment it requires the previous version so instead of applying the cumulative versions, we can only push the latest version. Also, the jar files will be copied to the T24 library folder when the updates get processed. Thus, new containers will automatically have the latest updates. This container must be connected to existing database so connection string needs to be fetched from kubernetes config map/secrets. The kubernetes definition file or helm chart should handle this part.

The custom script for deploying T24Updates and L3DSPackages is shown below. This generic script will be triggered regardless of whether the updates exist in the pre-image kit. However, based on the given conditions, the script execution will be controlled within the container. The same method can be scaled for any other components with similar deployment method.

In the automation approach, we need to trigger the script as soon as the container is started; therefore, the script needs to be executed in an entry point. Otherwise, we can execute the same with run time arguments. The first method is the far better option since it is a push method and we can handle the deployment issues gracefully.

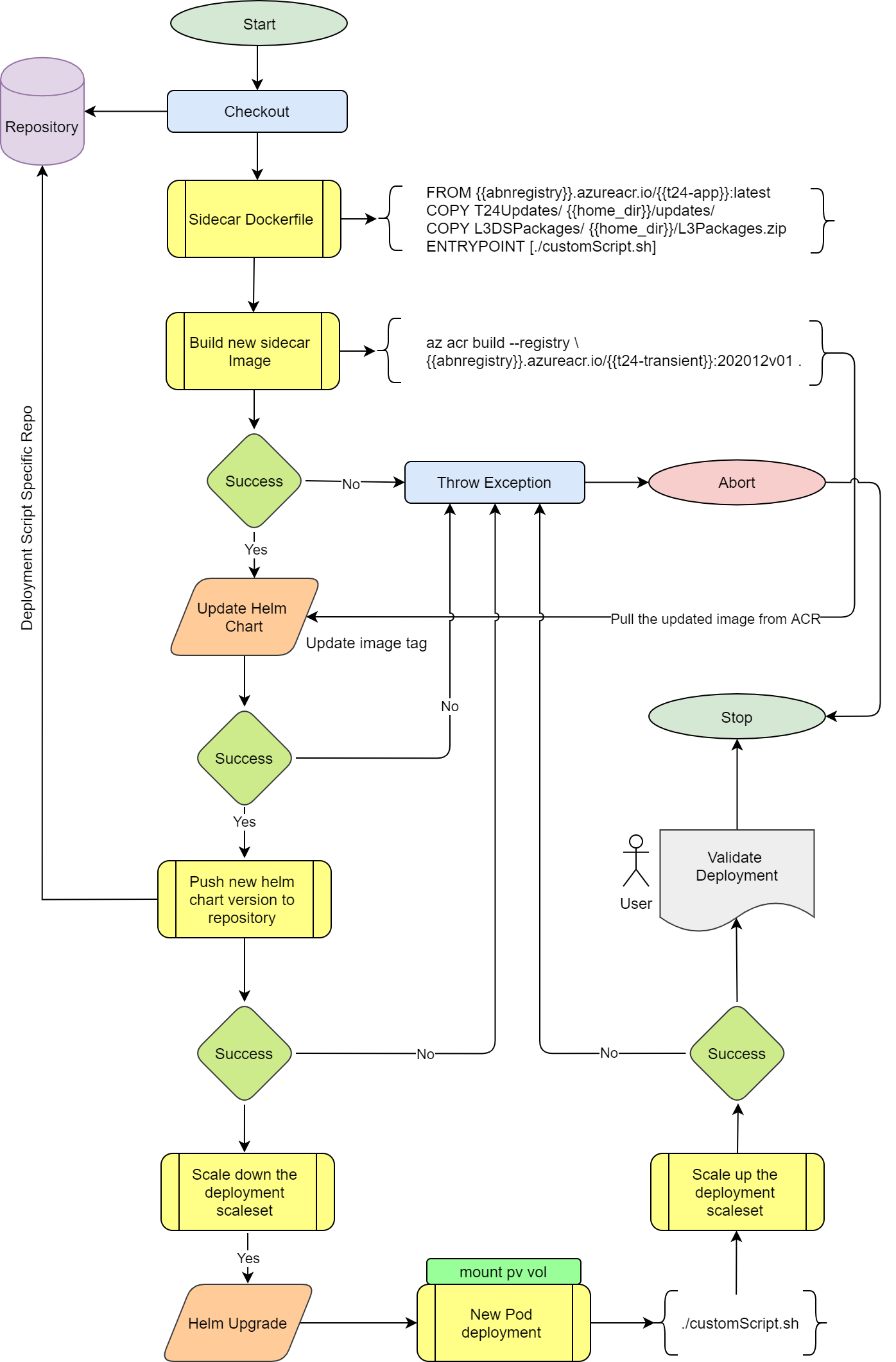


*Figure-1.9 customScript.sh*

### Process Flow

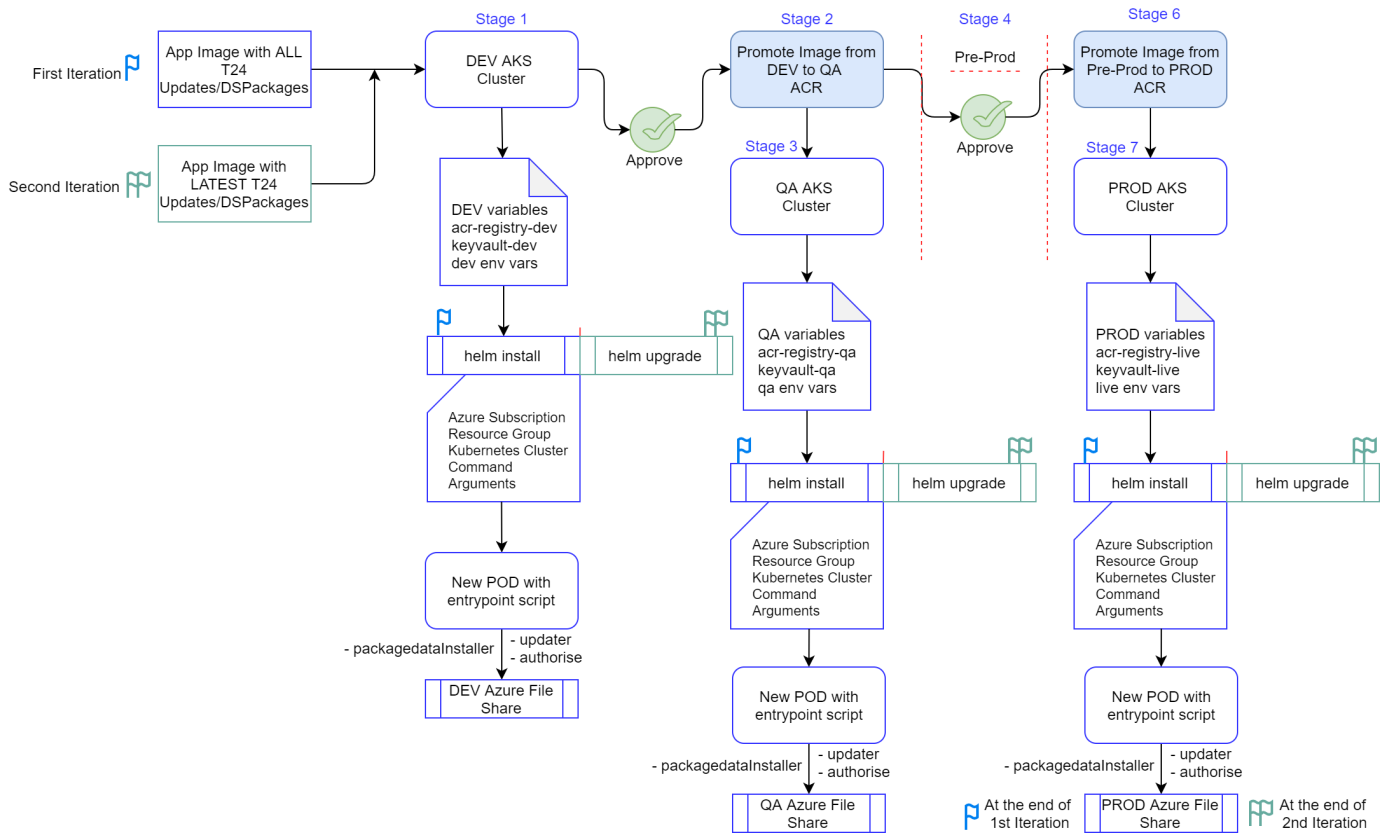
The following shows the automation steps involved in the T24 updates & L3 package deployments. All stages will be orchestrated from Azure DevOps dedicated pipelines.

1. Check out the deployment repository to get Helm Charts and other automation scripts into the Azure DevOps build agent
2. Build the new container image with sidecar Dockerfile. This image will contain the latest application binaries + T24updates/L3DSPackages
3. Update the helm chart with new image URL. And, update the helm chart version as well
4. Push the new helm chart to Azure repository to save the desired state configuration
5. Bring down the pod deployments scale set to 0
6. Once it is done. Execute helm upgrade command to apply the changes
7. Any minor changes in the deployment script will create a new pod altogether. *We may have to verify the downtime here.*
8. Ensure the new pod is mounted with PV volume.
9. Once the container started, it will execute our entry point script which will take care of the updates & package deployment
10. Bring up the pod deployments scale set to 1
11. The manual verification required once the deployment is complete.
12. Terminate the sidecar container & send the notification



*Figure-2.0 Approach I Process Flow Chart*

### T24/Packages Pipeline Stages



*Figure-2.1 Approach I Release stages*

## Approach-II

In this approach the container image will be built from the pre-image kit but before that, t24 libs from existing container will be copied to pre-image folder. There will not be any persistent volume in this approach; all the t24libs will be stored in the container file system.



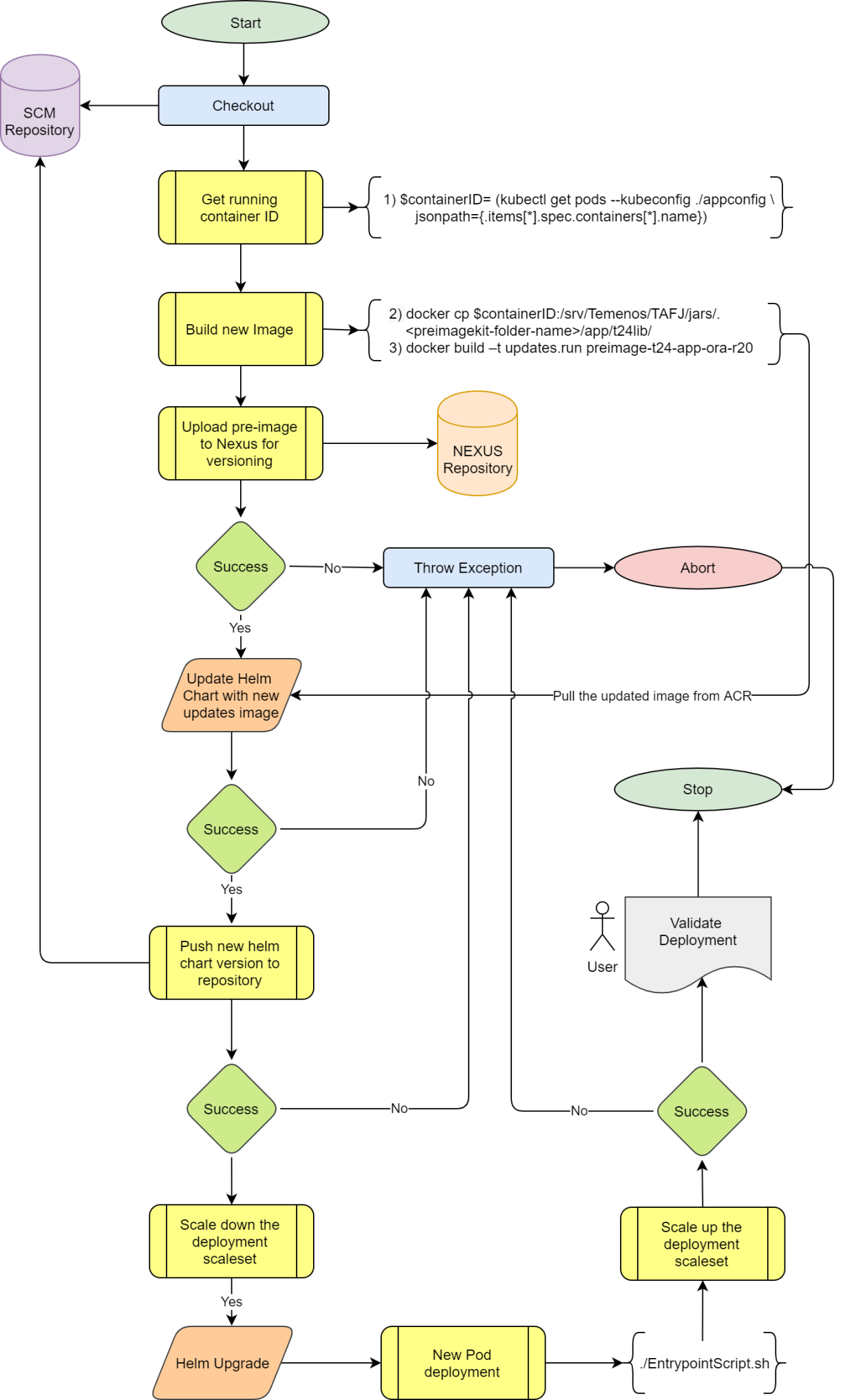
*Figure-2.2 Approach II Dockerfile*

We have to copy the t24lib/\*.jars from existing container into pre-image kit folder and build the container image from that. The same custom script will be used which is shown in the approach-I so the updates & packages will be processed once the container is started. Also, we have to ensure all pods are running with same image so the Transact helm charts needs to be replaced with new image as well. We have to upload the pre-image kit to Nexus repository for versioning purposes. For the subsequent deployment, the latest pre-image kit will be used.

### Process Flow

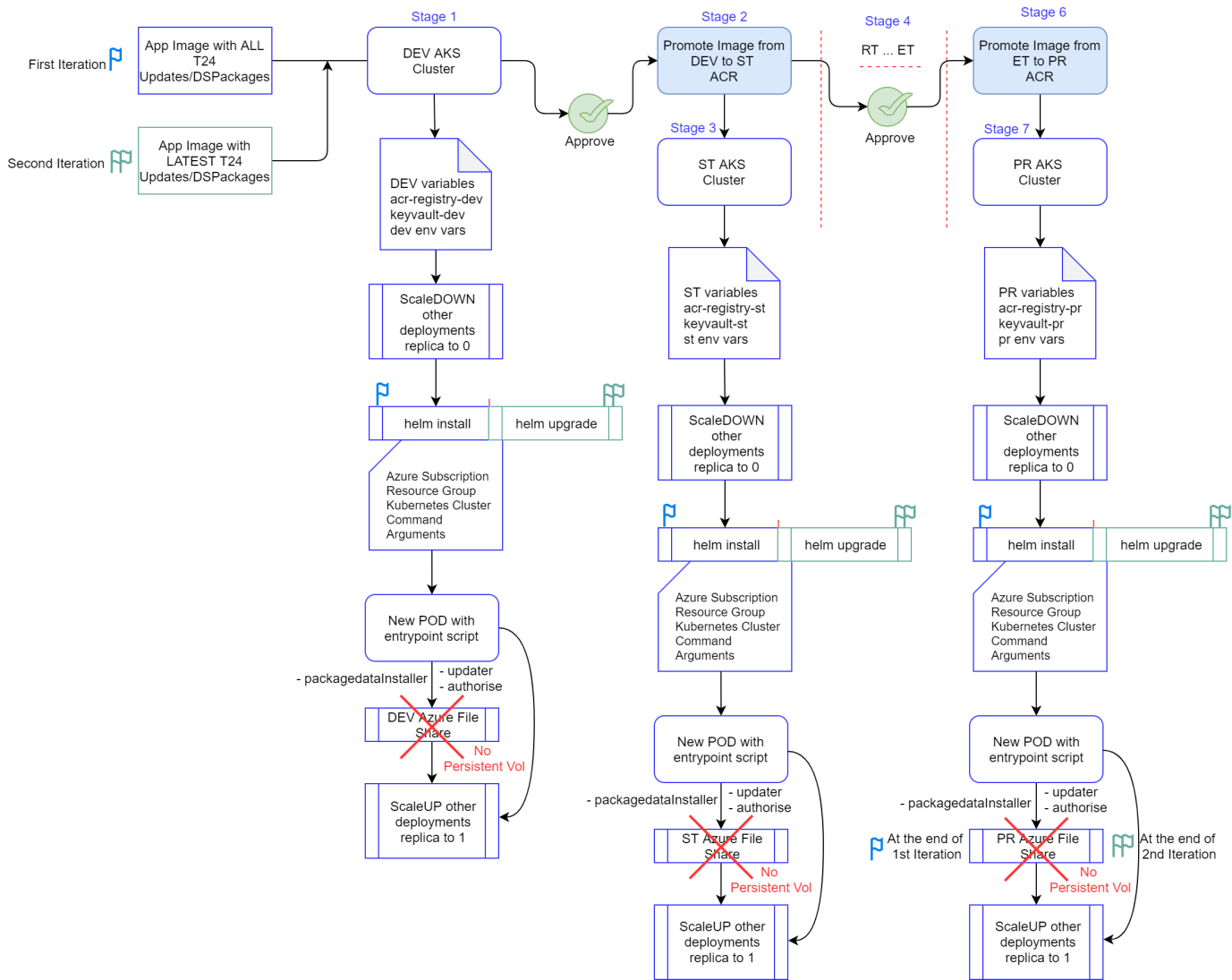
The following shows the automation steps involved in the T24 updates & L3 package deployments. All stages will be orchestrated from Azure DevOps dedicated pipelines.

1. Check out the deployment repository to get Helm Charts and other automation scripts into the Azure DevOps build agent
2. Get the container ID from the existing deployments
3. Copy the T24 libs from Docker container to pre-image kit
4. Build the new container image with above pre-image kit (containing updates & DS Packages).
5. Upload the pre-image kit to Nexus repository
6. Update the helm chart with new image URL. And, update the helm chart version as well
7. Push the new helm chart to Azure repository to save the desired state configuration
8. Bring down the pod deployments scale set to 0
9. Once it is done. Execute helm upgrade command to apply the changes
10. Any minor changes in the deployment script will create a new pod altogether. *We may have to verify the downtime here.*
11. Once the container started, it will execute our entry point script which will take care of the updates & package deployment
12. Bring up the pod deployments scale set to 1
13. The manual verification required once the deployment is complete.
14. Terminate the sidecar container & send the notification



*Figure-2.3 Approach II Process flow chart*

### T24/Packages Pipeline Stages



*Figure-2.2 Approach II Release stages*

# Challenges

* The CI/CD team have to adopt the Azure DevOps build and release pipeline configurations
* Components specific Azure repos needs to be maintained
* Integration of automation tests with Azure DevOps pipeline
* Maintenance of self-hosted agents\*
* Parallel jobs needs to be configured in Azure DevOps organizations
* Deployment of updates/packages to T24 without affecting other dependant pods
* Enable RBAC to control the user access
* Azure DevOps doesn’t support Nexus repository management via plugin so we have to develop custom script for uploading/downloading artefacts from repository

# Assumptions

* ABN Amro will procure the Azure DevOps license & Azure subscription
* Azure service principle will be used as an automation account
* The readiness of the repositories to build automation. All the application code moved into Azure DevOps Repos and Development team is comfortable using the same
* All the pre-requisites components build will be carried by existing pipelines
* The security gates/approval process in between the different stages can be customized as required
* If any of the artefacts are built manually then the source artefacts must be available/accessible to use it in the release pipeline
* Each environments will have dedicated AKS clusters and environment specific variables will be configured in the Azure variable group

# Considerations

## Availability

To monitor the application and report on issues, we can use Prometheus as a data source and the collected data will be sent to Grafana for visual dashboards. These toolsets will help us to monitor and troubleshoot any issues that may occur with the CI/CD pipeline. Azure monitor gives the standard logs; however, that may not enough for deeper analysis.

## Persistent Volume

The downtime is inevitable when rolling updates to the target, at the same time, the integrity of the platform also needs to be ensured to avoid any data loss. So it is highly recommended to use persistent volumes for all containers.

## Scalability

While configuring the AKS cluster, it is important to choose the number of worker node with auto-scaling. So that the Kubernetes master can scale out the number of nodes that runs our containers.

## Security

To minimize the security attack footprints, the remote access to the nodes must be restricted. And, if required, the client IP must be whitelisted with Network security group to allow inbound connections. Only public SSH key authentication is allowed for any servers within the whole system. Also, it is recommended to enable RBAC for the AKS cluster to avoid any security misconfigurations.

## Resiliency

Since AKS is a managed service, all the core services are resilient that monitor and restart the pods if any issues found. The Kubernetes schedulers can throttle the multiple pod deployments by allocating available nodes.