VIACOM

Continuous delivery pipeline and development workflow for containerized services and shared components

Setup procedure for end to end CI/CD workflow

Content platform initiative
Viacom Bridge

Content platform engineering team - Viacom
Darren Breed
David Sorbona
Raisel Melian

Mar-2017
v1.3-WIP RFC#2

Contents

Collaborators and revision

Authors

This document was created, updated, extended, improved and fixed by:

- David Sorbona
- Raisel Melian
- Darren Breed
- Tarique Rehman

Collaborators

The creation of this workflow and the procedure was possible thank the help of:

- Maxfield Burdge (Tools and Engineering support team)
- Carl Knoos (Multiplatform Cloud Services team - MCS)
- Kirk Beckford (Multiplatform Cloud Services team - MCS)
- Steven Azueta (Multiplatform/M&E team)
- Roman Shuhov (Multiplatform/M&E team)

Change log

Date	Version	Description
23-Mar-2017	0.1-wip	Document creation
05-apr-2017	1.0 -RFC#1	Version ready for RFC round 1
05-apr-2017	1.1 -RFC#1	Fixes
7-apr-2017	1.2-RFC#1	New step and updates in deploy plan to set tag “latest” in container registry
10-apr-2017	1.3-RFC#2	Add execution environment setup and updates to complete CD pipeline.

Abstract

Scope

This document describes all steps needed to setup an end-to-end agnostic continuous delivery workflow for the family of services developed under the “Viacom Bridge / Content Platform” initiatives.

The technology stack used is to achieve this is:

Topic	System
Source control management	Git
Source code repository	Bitbucket https://stash.mtvi.com
Continuous integration + testing + building + package	Bamboo https://bamboo.vmn.io
Container technology	Docker
Continuous deployment	Bamboo https://bamboo.vmn.io
Container images repository	AWS Elastic Container Repository (ECR)
Cloud orchestration layer	Jenkins http://jenkins.us-east-1.aws.cloud.viacom.com:8080
Execution environment	AWS ECS

In order to simplify the onboarding of new members and services the document is based in an example of a demo application developed in asp.net core.

The goal is to have a standardized continuous delivery workflow for any workload, language or technology stack that allow us to speed up the delivery of features to production.

Objective

The objectives of this document are:

- To understand the big picture of the continuous delivery workflow.
- To understand how to setup and manage local code repositories.
- To understand how to setup centralized code repositories in bitbucket.
- To understand how to setup a “building plan” in bamboo for continuous integration, testing, building and package.
- To understand how to setup a “Deploy project” in bamboo for continuous deployment.
- To understand the containerization style used on CI and CD.
- To understand how to setup container-images repositories in AWS Elastic Container Repository (ECR)
- To understand how to setup environments in AWS EC2.

Requirements

The requirements needed are:

- Basic knowledge of distributed source control management like “git” or “mercurial” (hg)
- Basic knowledge of CI/CD.
- Viacom LDAP account that belongs to the “Crowd security group”

  “content_platform_engineering”

- Access to create code repositories in bitbucket: https://stash.mtvi.com
- Access to create build and deploy plans in bamboo https://bamboo.vmn.io
- Access to create container registries and other AWS resources. In this example, for this you have to belong to the AD group: “DL_ContentPlatform_Engineering”

Note:

- Bitbucket and Bamboo access can be requested to the “Tools and Engineering support” team thru jira.
- Changes on “Crowd security groups” are managed by the “Tools and engineering support” team thru jira.

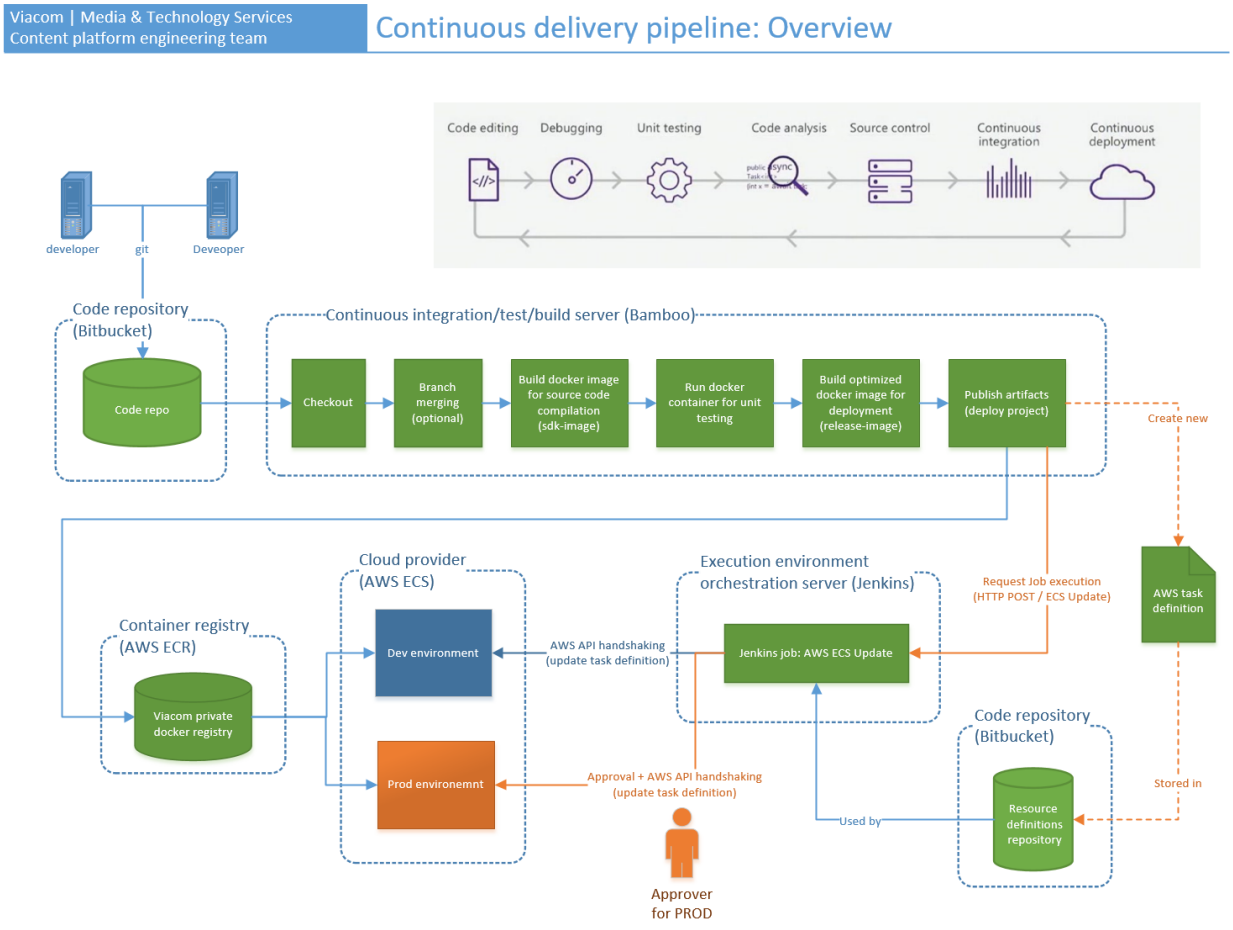
Continuous delivery pipeline

The main idea behind the concept of “continuous delivery” is to establish a procedure to deploy changes in production as fast as possible in a consistent and safe way, in order to reduce down-times, avoid unexpected errors and have happy users.

We are going to call this as the “Viacom continuous delivery workflow”

Overview

The following diagram shows how the pipeline looks like and the most important steps:



The complete pipeline has more steps but generally speaking we can say that the process looks like this:

1. A “changeset” is pushed to the central code repository thru git.
2. The integration server (bamboo) is watching the central code repository, and will trigger a build (bamboo build plan) defined previously.

3. The build plan will:

    a. Checkout code from repository into the “bamboo build plan working directory”.

    Good to know

    The specific branch used during the checkout is defined in the “build plan setup”. By creating “branch plans” is possible to execute the same “build plan” (the same build steps) but using another branch during the checkout. You will see this in action later in this document.

    b. Based on the branch plan configuration, a “changeset” could be integrated into another branch, for example the “development branch”. If the code merge works as expected the pipeline will continue running.

    c. The merged code is built and tested. If the tests are ok the pipeline will continue.

    d. The code will be “containerized” into a docker-image, this will be the “artifact” produced by the pipeline.

Requirements and restrictions

In order to onboard your project into the VCDW (“Viacom continuous delivery workflow”) the following requirements and restrictions have to be fulfilled:

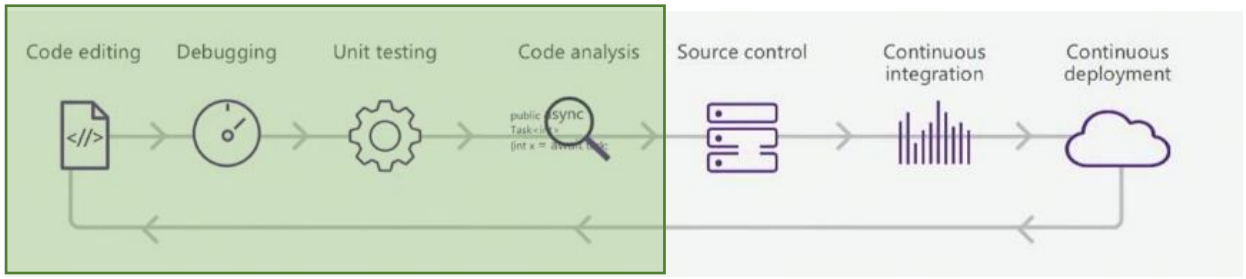
- Your code has to run in a docker Linux container.
- You have to provide a public or custom “sdk-container-image” with all the dependencies needed to build your project including: package restoring, compression, string obfuscation, CSS compilation, etc. This image can be hosted in a public registry like docker-hub (recommended) or in a private registry like the Viacom AWS ECR instance.
- You have to provide a public or custom “optimized-container-image” that will be used in runtime on the execution environment (in this example, AWS ECS)
- You project must produce an artifact to be deployed. For “executable” artifacts, like a web service, will be a stand-alone docker-container image ready to be deployed. In the execution environment, it shouldn’t need any other external dependency. For “non-executable” artifacts, like a ng packages, the “containerization” is not mandatory if you will publish it to a package repository like nuget.
- The result of the unit-tests has to be saved as a .xml file in any of the format supported by the current “Bamboo test parsers plug-ins” available in the current Viacom’s Bamboo instance. In the following sections, you will find the list of the current supported plug-ins.

Out of scope in this version

- Code coverage analysis won't be part of the build plan. It will be included in the future.
- This version will only support a simplified subset of the git-flow branch management strategy.
- AWS Cloud formation utilization will be covered in a future version.
- Deployment on AWS PROD environment.

Development workflow

The complete workflow includes more steps and could have additional steps base on particular situations, but as an introduction we can define the following standardized development workflow:



General procedure

Let's imagine that a user needs to add a new feature to a project that is managed by the "Viacom continuous delivery workflow", so the steps would be the following:

1. The developer clones a code repository from bitbucket to its local machine. If you already have a local repository make sure you have the latest changests and you working directory is in the correct branch.

```
git checkout [branch name you are working]

git pull
```

2. The developer creates a new "feature branch", for example like this:

```
git branch feature/NewValidationsForOrderProcessing

git checkout feature/NewValidationsForOrderProcessing
```

WARNING

Branch naming convention is really important because the Bamboo integration server uses the branch-name to apply automations. Therefore, in this case the prefix "**feature/**" will command Bamboo to integrate this branch into the "development branch" after a changeset of the "feature/" branch is pushed to bitbucket.

3. The developer codes something awesome and the unit-tests needed to cover the new code lines.

TBD: minimum code coverage accepted level is not defined yet.

4. The developer debugs, introduce fixes into the code base, do refactoring and runs unit-tests in a loop until all unit-tests are passed and the code quality is good enough.

IMPORTANT

Having all unit-tests passed in green is mandatory for a “feature/” branch, but this is not a guarantee that the code will behave as expected. It is very important to apply common sense and avoid pushing broken code to the main code repository. This is critical if you enable automatic integration into the “development branch”.

If you are working in a personal or experimental branch, you are allowed to push code into the main code repository just for backup and sharing purposes even if the unit-tests fails or the code is broken or incomplete. In this scenario is assumed that you are working to reach the “all green” state.

5. The developer commits its changes. Here a tip: do commit regularly whilst you are coding, thus you will have a returning point in case your coding trip goes wrong.

6. The developer pushes committed changes to the main repo (remote/origin) when quality attributes are met and unit-tests are passed.

Development and integration strategies

The general procedure is a simplification of how you should work in real life. We suggest using any of the following of strategies for you daily development workflow. Each strategy is tied to the way you integrate the code into high level branches.

Personal/experimental branch strategy

In this strategy, you decided to enabled the automatic branch integration in the Bamboo CI pipeline, so any change in a “feature/” branch will be automatically integrated into the “development” branch.

But sometimes you can’t push code to the main repo because it is not complete or there are some unit-test failing. But at the same time, you want to backup it. To resolve this, you create a “personal/experimental” branch that eventually will be integrated into the “feature/” branch thru a pull request or thru a direct merge via git.

Under this scenario, the development workflow will look like this:

1. The developer creates a “personal/” branch from the “feature/” branch and checkout to that branch. The integration chain will look like this.

davidsorbona/feature/A → feature/A → development → mater

2. He writes code that looks like a Beethoven symphony but it is still a work in progress and several unit tests fails.

3. He pushes his code to main repository to keep the code backed-up, and go home. Note: Bamboo branch plans are not triggered since Bamboo does not recognize the “davidsorbona/” prefix.

4. Next day, the developer completes its master piece.

5. He starts the debug, fix, unit-test loop until all unit test are passed and code quality is good.

   Decision point – Option A

   - He creates a "pull request". The pull request is a way to say "Hey QA team, could you please review and integrate my code into the "feature/" branch?"

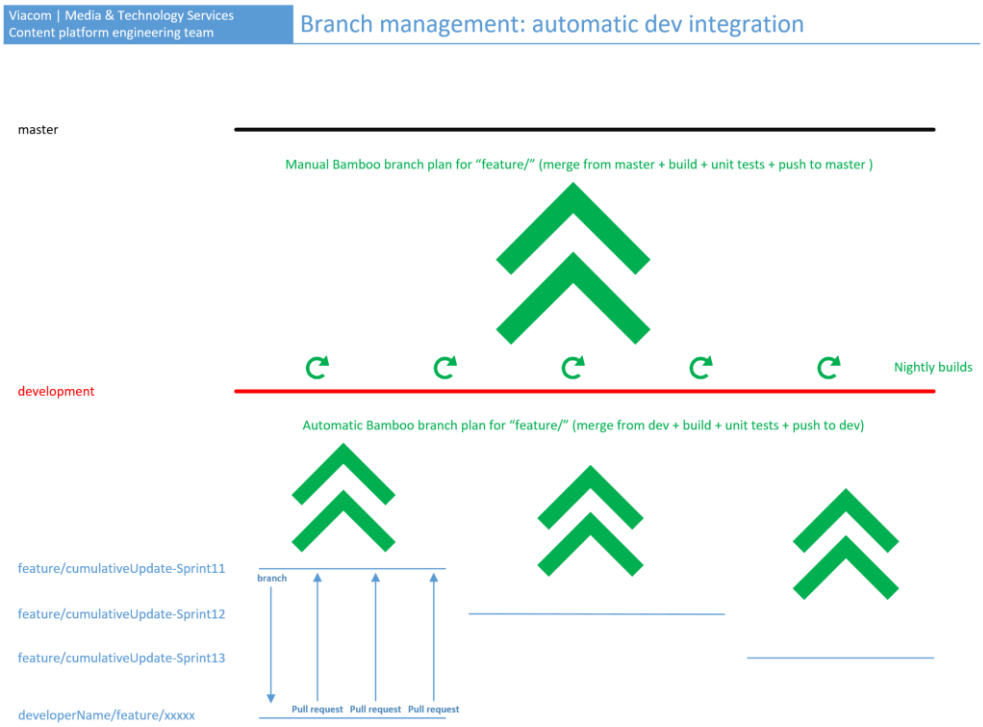
   - The QA team or tech lead review your code, if good it is approved and integrated into the development branch. Otherwise it is rejected and you will receive instructions to introduce fixes and try again later with another pull request.

   Decision point – Option B

   NOTE: This option is recommended if you are the only one working in that repository to avoid adding an administrative overhead that does not make sense.

6. He merges his code from your "personal" branch into the feature branch.

Here an overview of how the workflow looks like:



Pull requests strategy

When several developers work in one code base, it is a good practice that someone review the code before integrating it into a high-level branch like "development" or even into a "feature/" branch. That

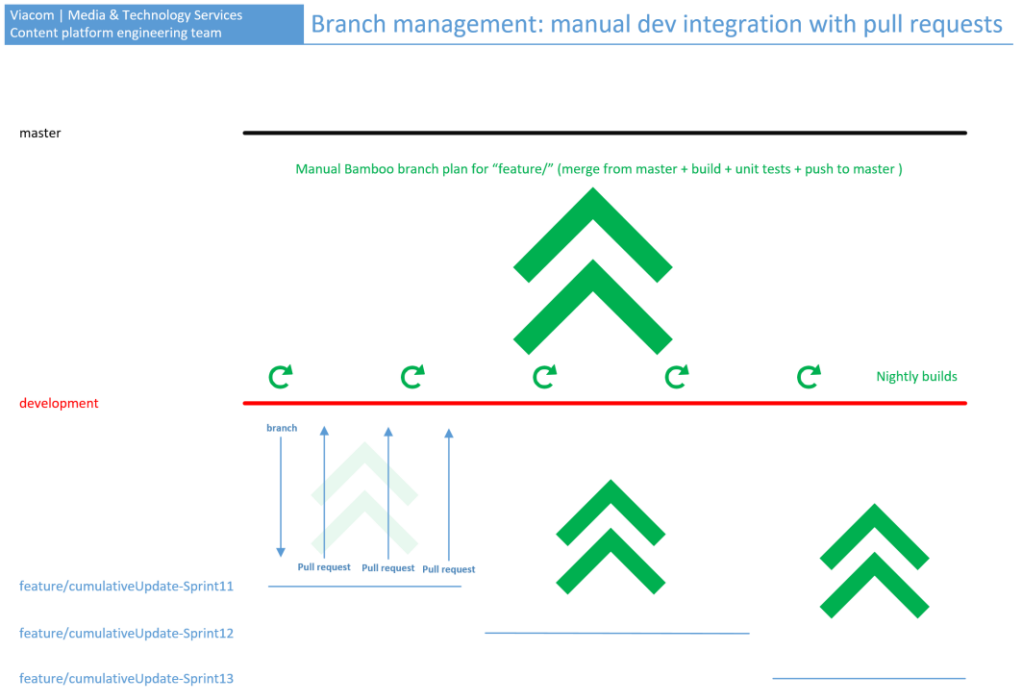
can be accomplish by implementing “pull requests” integration strategy in your development workflow. This is enabled from the main code repository in bitbucket.

When you work with pull requests someone has to integrate the code into the “development branch”, it means that the Bamboo automatic branch integration thru plan branches have to be disabled.

Under this scenario, the development workflow will look like this:

1. The developer creates a “personal” or “experimental” or “feature” branch. Note: in this strategy the branch naming convention won’t have any impact at bamboo side.
2. He writes some fantastic code but it is still a work in progress and several unit tests fails.
3. He pushes his code to main repository to keep the code backed-up, and go home.
4. Next day, the developer completes its master piece.
5. He starts the debug, fix, unit-test loop until all unit test are passed and code quality is good.
6. He creates a “pull request”. The pull request is a way to say “Hey QA team, could you please review and integrate my code into the development branch?”
7. The QA team or tech lead review your code, if good it is approved and integrated into the development branch. Otherwise it is rejected and you will receive instructions to introduce fixes and try again later with another pull request.

Here an overview of how the workflow looks like:



Good practices to avoid merge conflicts

The best way to resolve a merge conflict is avoiding it to happen. Here some ideas:

- Make sure that your local working directory is up-to-date.
- Make sure you are working in the correct branch.
- 1 type = 1 file rule: create each type (class, enun, struct, etc) in its own file. Don't be shy, files are free. The more files you have the less conflicts you will get.
- Always add code lines at the end of the file. Avoid adding or removing lines in the middle of the file when is not necessary.
- Two or more people working in the same class? partial classes could help you.
- Make sure that you git "ignore" file is present in your working directory.

  You can find "ignore" files for almost any kind of project here:

  https://help.github.com/articles/ignoring-files/

  Do you need more help with ignore files? Go here -> https://help.github.com/articles/ignoring-files/

Code quality attributes

There are tons of bibliography that describes what is code quality. We expect that code quality attributes are met before it is integrated into high level branches (e.g. development branch, release branch or master branch).

Quality is not a thing that can be introduced at the end, you have to try to keep your code “good” and “clean” from start.

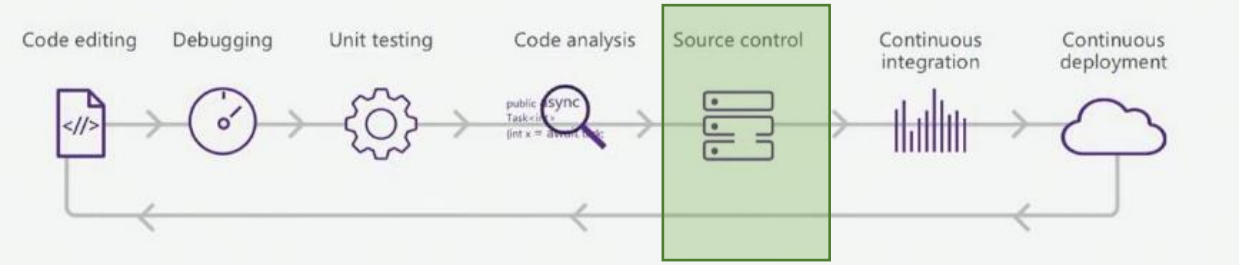
As a minimum base-line we define the following attributes that defines what is “good” code:

- Avoid one big all-in-one class.
- Use class inheritance carefully.
- Explicit and very clear names for parameters, variables, members and types even if the name looks too long. Think this in this way, if you read a name and you have to ask someone what it means, then the name is wrong.
- Comment when is needed, but keep in mind that good code does not need comments.
- Use dependency injection when makes sense to use it.
- Apply design patterns, avoid reinventing the wheel.
- The classic: keep cohesion high and coupling low.
- Make explicit dependencies in constructors or methods. Do not hide dependencies inside the body of a method or function. Decouple those dependencies by using abstractions.
- Make your code easy to test.
- Keep cyclomatic complexity low. https://en.wikipedia.org/wiki/Cyclomatic_complexity
- Indent your code. Make it easy to read.

> “*Everyone writes codes that can be read by computers but not anyone can write code that can be read by humans.*” Martin Fowler.

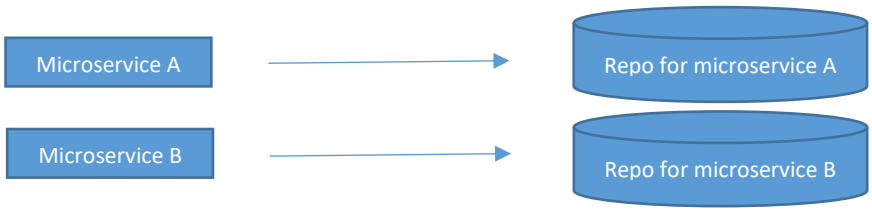
Source code management

Introduction



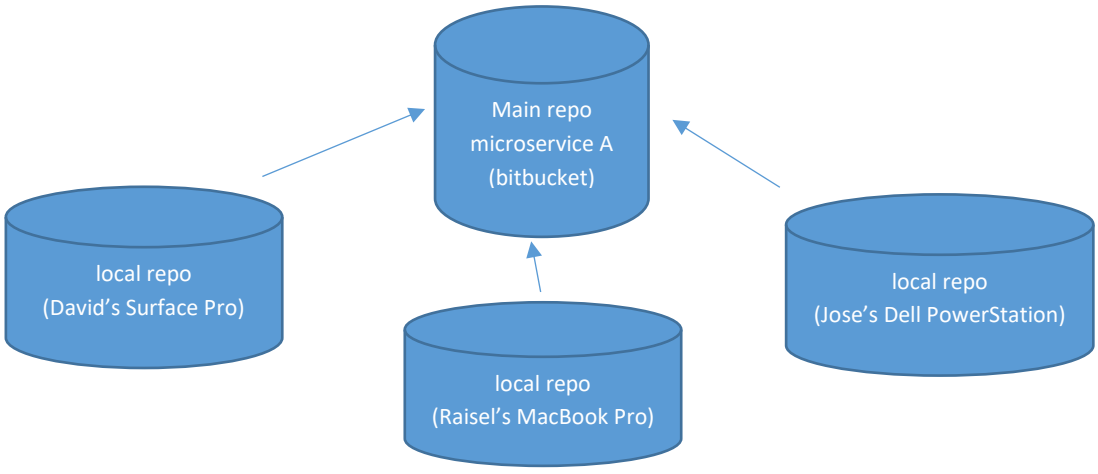
Source code management is the key part upon is built the continuous delivery workflow. In this document, all examples are described in “git”, but it could be adapted in order to use other SCMs tools like mercurial (hg).

The source code of each "deployment unit" (e.g.: website, microservice, etc.) must be stored in an individual code repository hosted in the Viacom's main code repository system. This procedure defines bitbucket (aka stash) as the main code repository system.



Code repository in server

Git is a distributed code management system, it means that there could be different repositories linked together distributed geographically. To keep it simple we will use one “main repository” (aka in git terms as “remote/origin”) hosted in bitbucket and many “local repositories” hosted in the developer machines.



So, David, Raisel and Jose are working together in the same microservice A.

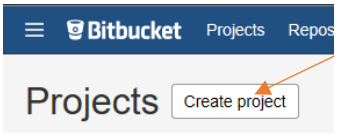
IMPORTANT

Developers should have only those local repositories needed to complete its tasks. No more, no less.

Repository setup procedure

The steps to create a repository in bitbucket (aka stash) are the following:

1. Using your LDAP account log into Viacom bitbucket portal: https://stash.mtvi.com
2. Optional step: create a project called “Viacom Content Platform” to group your repositories.



3. Create a repository

Code repository naming rules

system code + development stack + project name

- all lowercase.
- [system code]
    - 3 letters max.
    - mandatory: "vcp” -> Viacom Content Platform
- [development stack]
    - java
    - netcore
    - netfx
    - npm
    - phyton
    - Others to be defined.
- [Project name] defined by architecture/leadership team

Example: *vcp-netcore-webappdem01*

4. Done! You have a new code repository and a URL link like this:

https://stash.mtvi.com/projects/VCP/repos/vcp-netcore-webappdem01

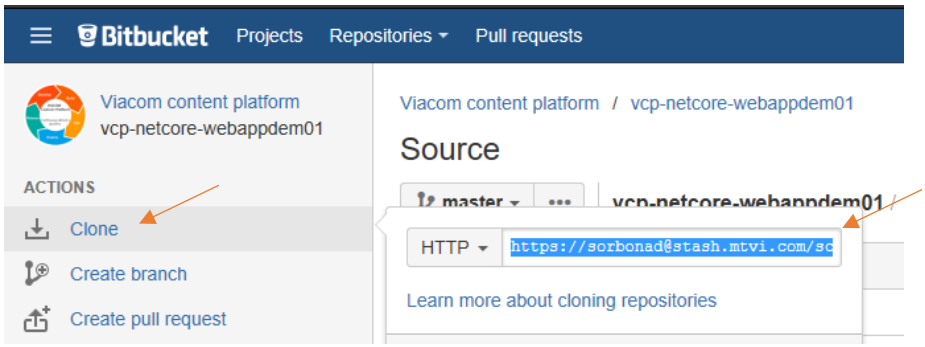
Code repository setup in development unit

The are several ways of doing this, avoid a head-ache and follow this safe procedure:

1. Create a local directory to host the repository. For this example, I have a folder called:

   "c:\vimn-america_code-repository\Viacom.ContentPlatform"

2. Go to the bitbucket repository portal and get the repository url:



3. Open a command line shell and run these commands:

   cd [path where you are going to host the repository]

   git clone [url you get from bitbucket in step 2]

   Example

   cd c:\vimn-america_code-repository\Viacom.ContentPlatform

   git clone https://sorbonad@stash.mtvi.com/scm/vcp/vcp-netcore-webappdem01.git

4. Done! Your local repo is ready to start coding.

Committing and pushing changes

All code, artifacts, configuration files, project settings, unit tests, etc. must be placed inside the "working directory", in this case:

c:\vimn-america_code-repository\Viacom.ContentPlatform\vcp-netcore-webappdem01

WARNING

In this example I am creating an application from scratch, but if you are start working in an existent code base you have to start coding in the correct branch.

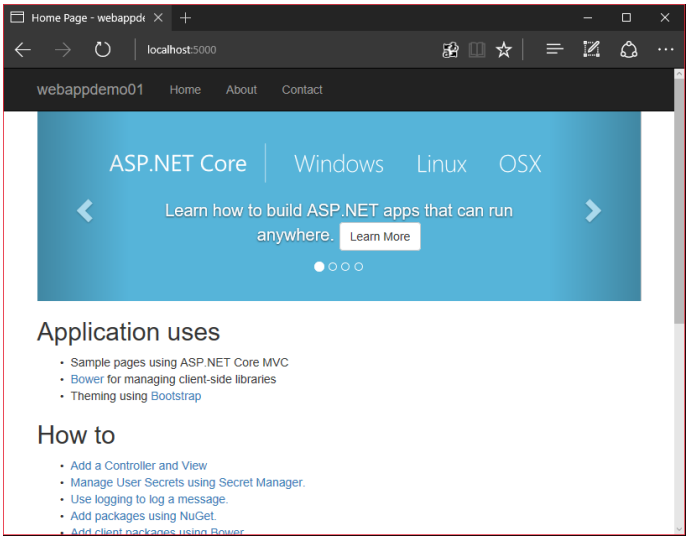
For changing branches, you just have to run this command:

```
git checkout [branch name]
```

To continue with the example, I will create a .net core mvc web app by running these commands:

```
cd .\vcp-netcore-webappdem01\

dotnet new mvc

dotnet restore

dotnet run
```

Open http://localhost:5000 and you should see this:



The previous commands have created all the files needed to execute the application under the "working directory" and under the "master" branch.

The code, track, commit and push loop

First make sure you have a correct "*.gitignore"* file in the root of the working directory, otherwise you will track files that are not needed.

The track, commit and push loop procedure looks like this:

1. Tell git to track all working directory file system changes (new files, deleted files, modified files, etc)

   ```
   git add .
   ```

2. Commit your changes and add a good comment.

   ```
   git commit –m "created all needed files to execute this application"
   ```

3. Have you run unit tests and all are green? If yes, continue, if no stop and make that happen first.

4. Now push this changeset to bitbucket (origin/master branch)

   ```
   git push
   ```

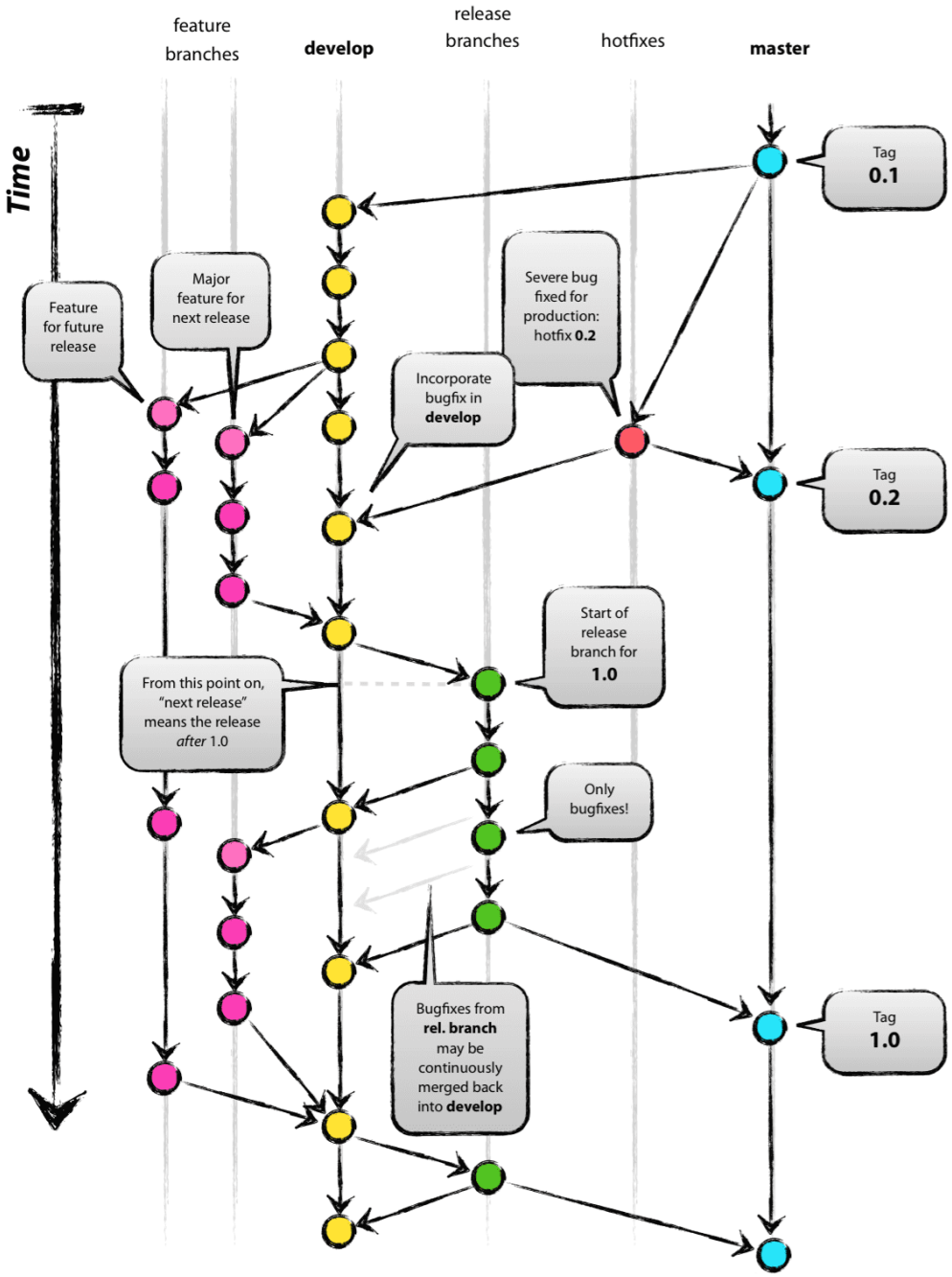
5. Done! Now your changeset will be hosted in bitbucket

6. Code something else, go to point 1.

Branch management

THIS SECTIONO IS A WORK IN PROGRESS

The Viacom continuous delivery workflow definitive branching style is still in development but it definitely will be something similar to git-flow http://nvie.com/posts/a-successful-git-branching-model/

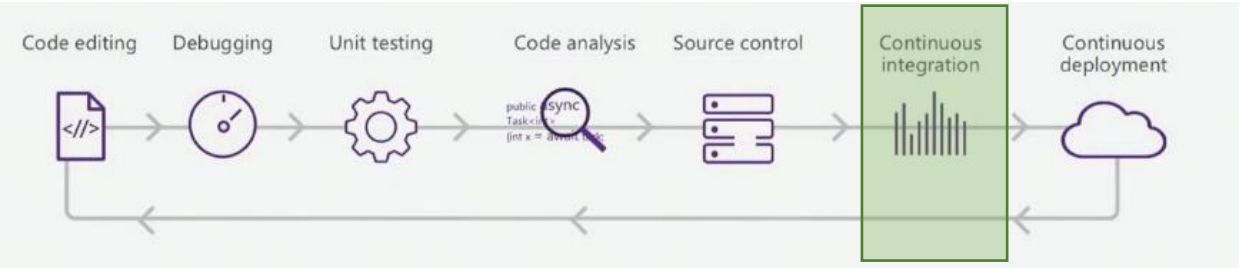


Currently the Viacom CD workflow supports four branches:

Branch	Description
master	• Host the code that is in production. • You never develop this branch. • There is only 1. • Name: master
development	• Integration branch where all changes are merged before going to master. • You never develop this branch. • There is only 1. • Name: development.
feature branches	• Branch where you code something. • There could be several feature branches at the same time. • Once a commit from a feature branches is pushed, it is merge to the development branch • Naming convention: mandatory naming prefix: “feature/”
Experimental or personal branches	• Where you can do experiments, play with the code, break things • There could be several experimental-personal branches. • Naming convention: mandatory naming prefix: “userName/” or “experimental/” • This branch can be pushed to main central repository for backup purposes.

Continuous integration pipeline

Introduction



Each code repository will be built by a continuous integration pipeline called “Build plan” that have to be setup in the Viacom integration service: Bamboo.

The containerized-approach put some restrictions in the way we can work with bamboo, I will explain every decision below. “Containerized-approach” means that the build, unit testing and other activities will happen inside a container running in a bamboo agent of the bamboo cluster.

As a high-level overview, let’s say that every “Bamboo build plan” will trigger only 1 “job” with only 1 “stage” with several sequential “tasks”.

The tasks sequence was defined to be used under any workload, language and technology stack; It just needs minor tweaks. The most important steps are:

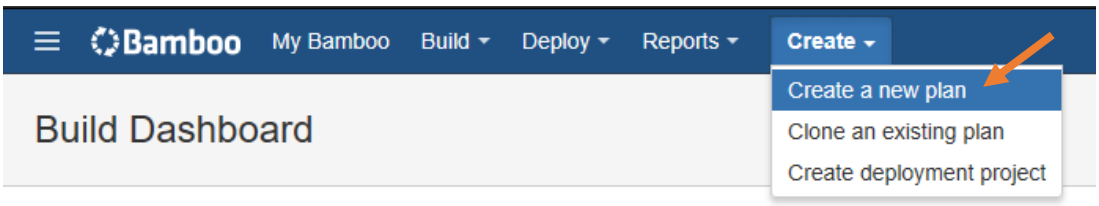
1. Code check out.
2. Optionally, bamboo will merge the “feature/*” branch into the “development” branch.
3. Build a sdk-container-image using as base another container-image that includes the SDK and all dependencies needed to build the application.

   NOTE: This can be a public image hosted for example in docker-hub or a custom image hosted in the AWS Viacom’s private repository
4. Run unit testing in a the sdk-container.
5. Parsing unit test results using bamboo plug-ins.
6. Build an optimized container-image (“release-image”).
7. Save the release-container-image as a “build artifact”. A build artifact is the result of the build process.
8. A little bit of clean up.

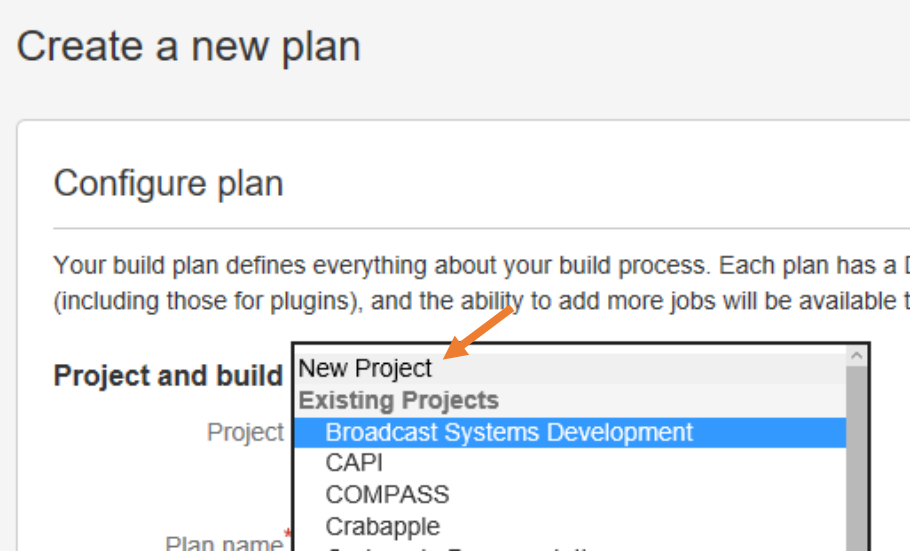
Build plan setup

In order to create a new “Build plan” in the integration server (Bamboo) follow the next steps:

1. Log into bamboo using your LDAP account: https://bamboo.vmn.io
2. Create a new plan



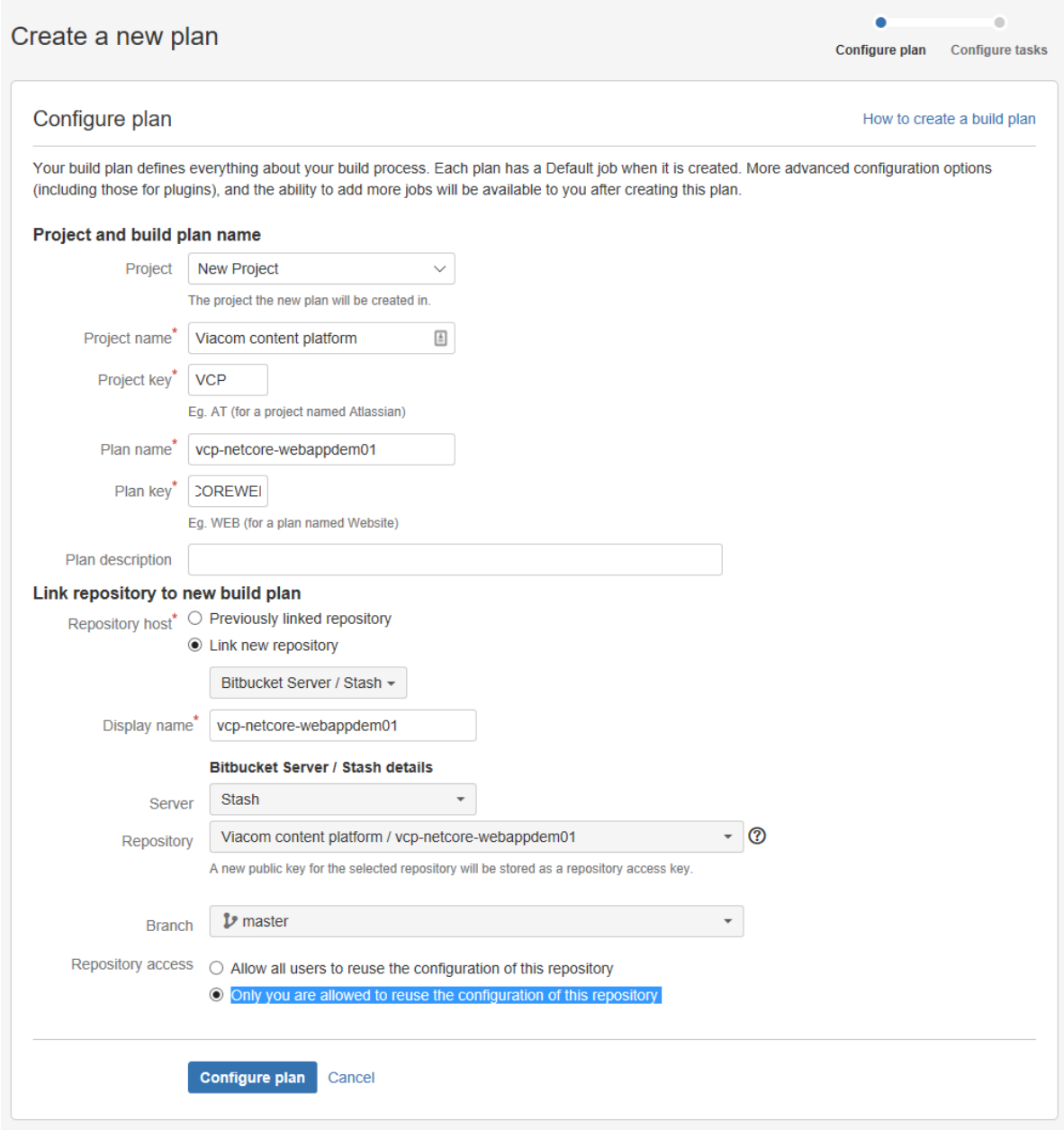
3. Decision point: you can create your plan under a previous “Build project” or you can create a new one. For this example, I will create a new project so select “New project”



4. Complete the form with the following information:

- **Project name**: “Viacom content platform”
- **Project key:** “VCP”
- **Plan name:** same as your repository name, in this case “vcp-netcore-webappdem01”
- **Plan key:** same as the repository name but without “-” and all uppercase
- **Repository host: “**Link new repository” and then select “Bitbucket Server / Stash”
- **Display name:** same as the repository, in this case “vcp-netcore-webappdem01”

- **Server:** “Stash”
- **Repository:** select your repository, in this case “Viacom content platform / vcp-netcore-webappdem01”
- **Branch:** “master” (don’t worry, we will manage other branches of the same repo thru something called “Branch plans”)
- **Repository access:** “Only you are allowed to reuse the configuration of this repository”



5. Click on “Configure plan”. Done! you have the wrapper to host jobs and tasks. See next section.

Configure tasks

The wizard will ask you to create the “default job” by setting up its tasks. If a build plan has multiple jobs then they can run in parallel, but for this example and to keep it simple we are going to use only 1 job, the default one.

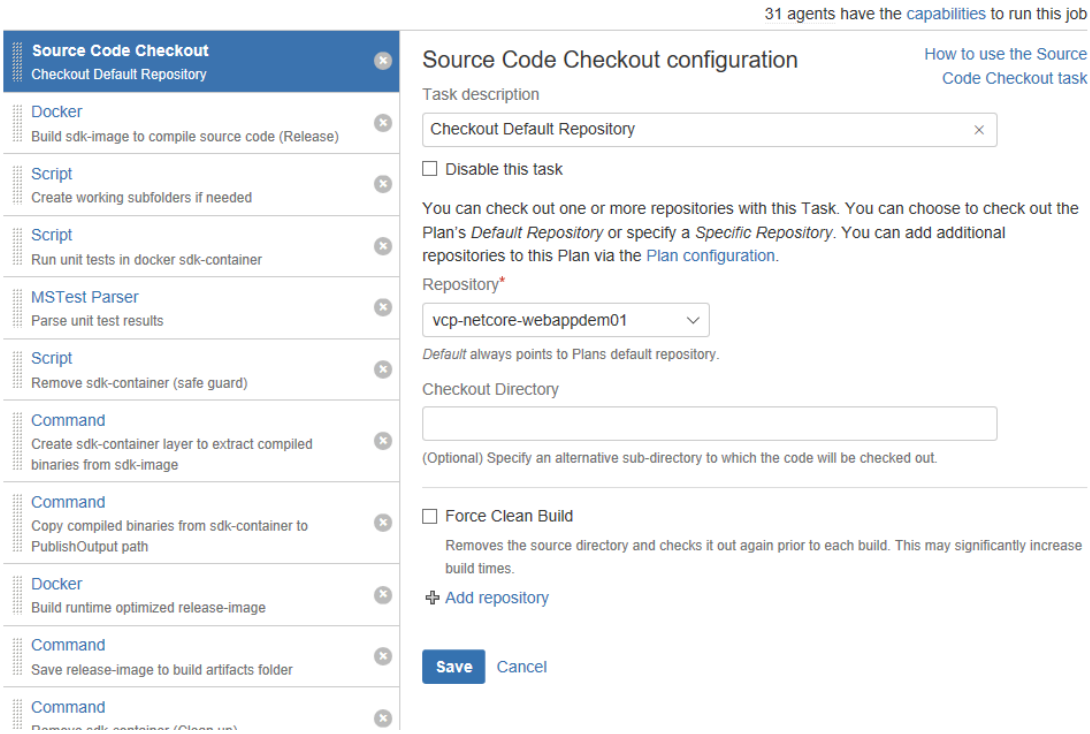
NOTE

I think that having multiple jobs in parallels makes more sense if you are building multiple artifacts that are part of the same release. For now, we are not taking that approach so one build plan will produce one artifact to be released during deploy step.

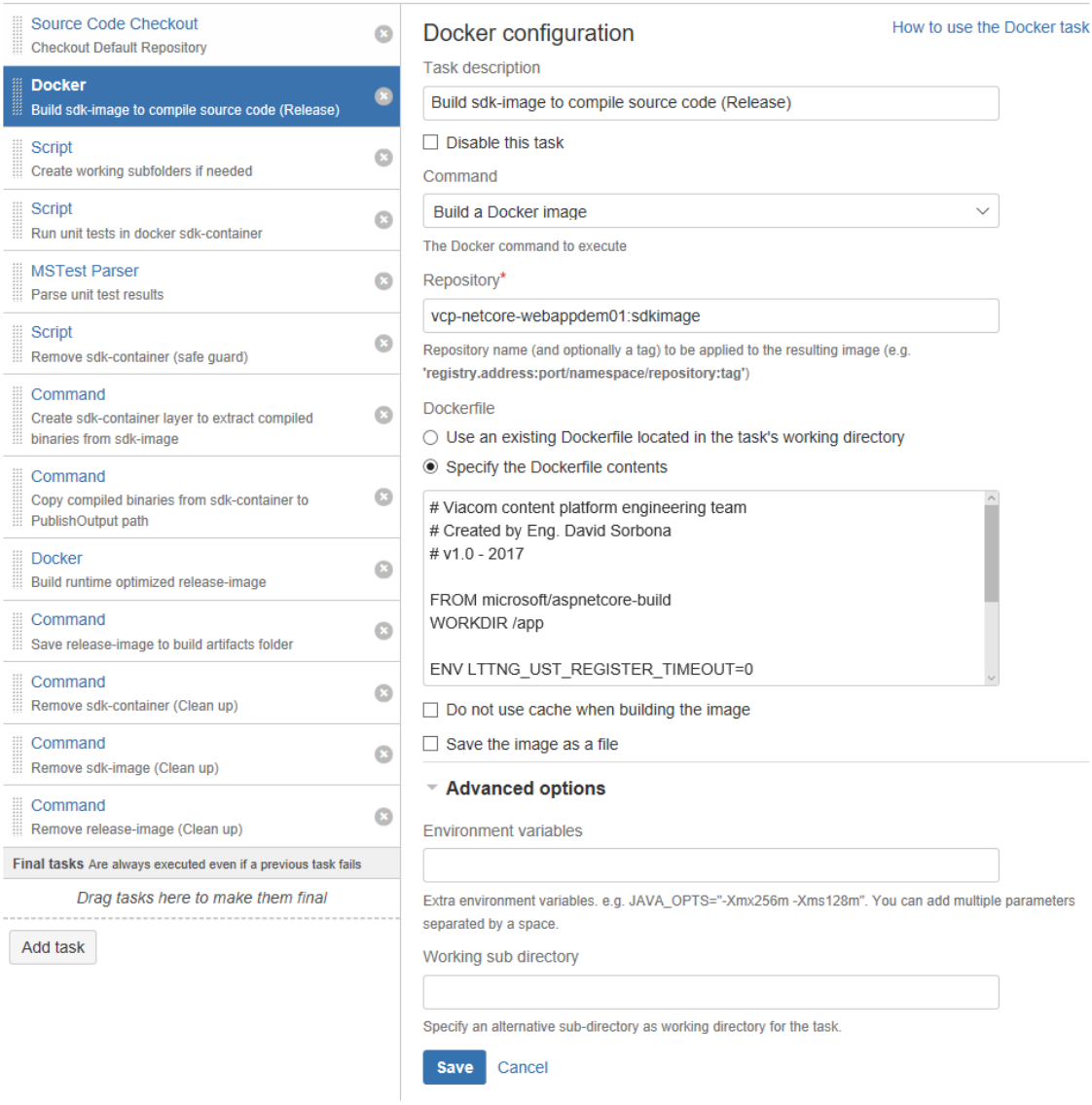
Basically, we are going to execute a sequence of different type of tasks that will transform the source-code into an “optimized” Docker container-image ready to be deployed, this will be the “release-artifact”.

Below you will find the details of the tasks to be created. Please keep in mind that the order is important, don’t change it. These tasks will be executed in sequence:

1. Task: “Source Code Checkout”



2. Task: “Docker / Build sdk-image to compile source code (Release)”



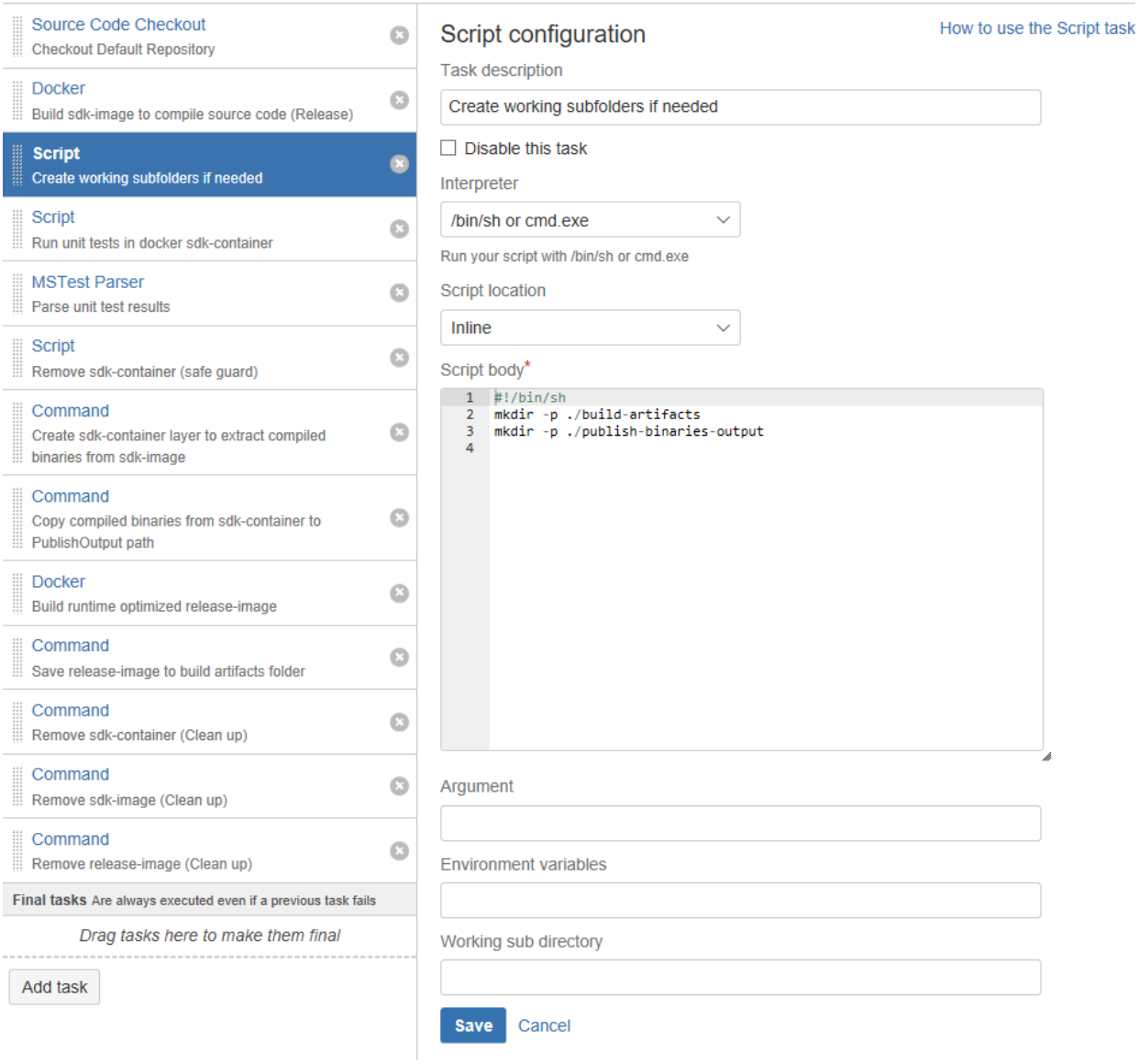
Docker file content:

```
FROM microsoft/aspnetcore-build
WORKDIR /app
ENV LTTNG_UST_REGISTER_TIMEOUT=0
COPY . .
RUN dotnet restore
RUN dotnet publish /app/vcp-netcore-webappdem01.csproj --output /out/ --configuration Release
```

NOTE: the source Docker image must have all dependencies needed to build the source code including package restore, text compression, obfuscate strings, etc.

Task result: binaries and a “SDK Docker image”

3. Task: “Script / Create working subfolders if needed”

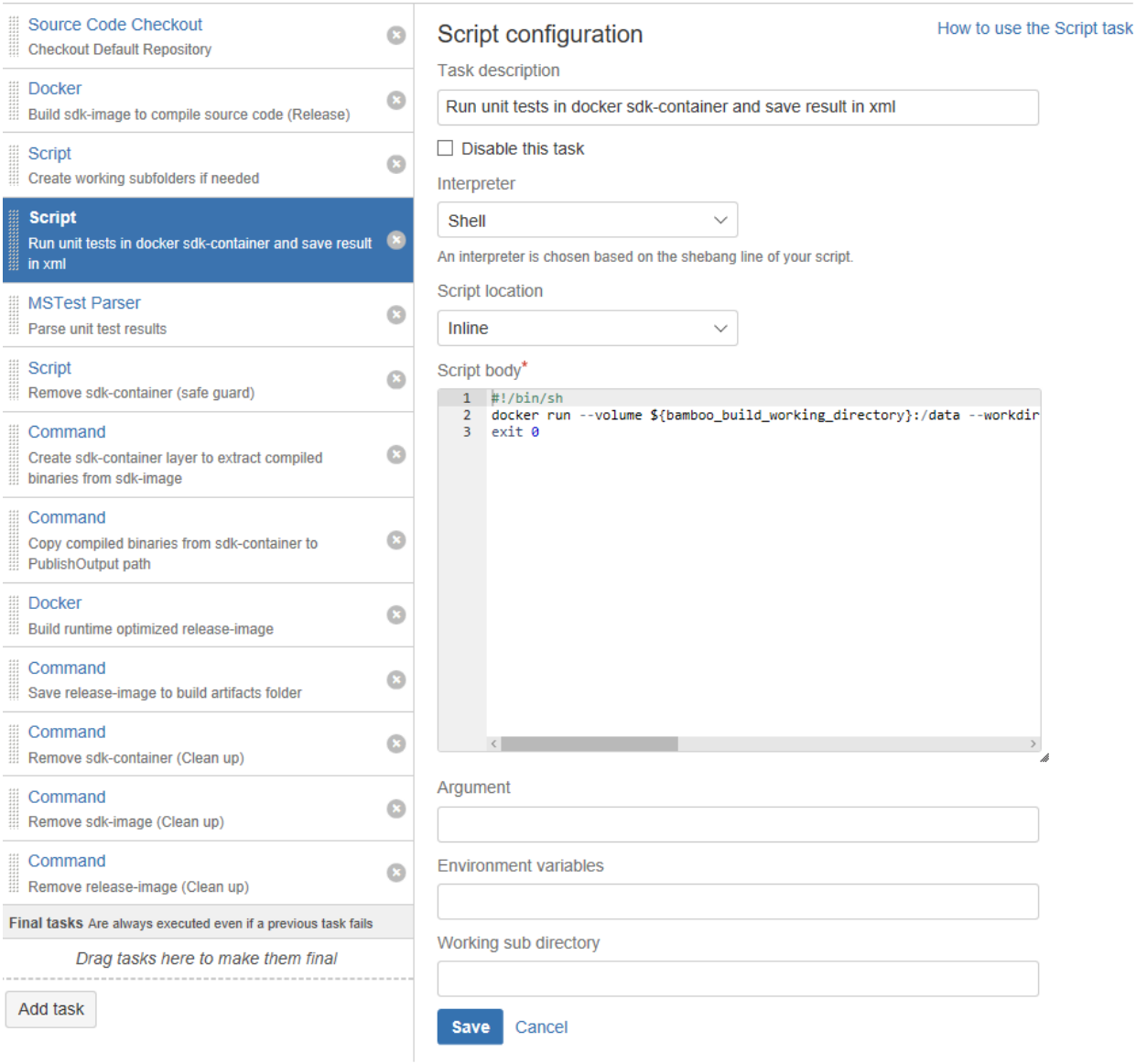


Script body:

```
#!/bin/sh
mkdir -p ./build-artifacts
mkdir -p ./publish-binaries-output
```

Task result: temporal folders used to store files during the CI pipeline execution.

4. Task: “Script / Run unit tests in docker sdk-container and save result in xml”



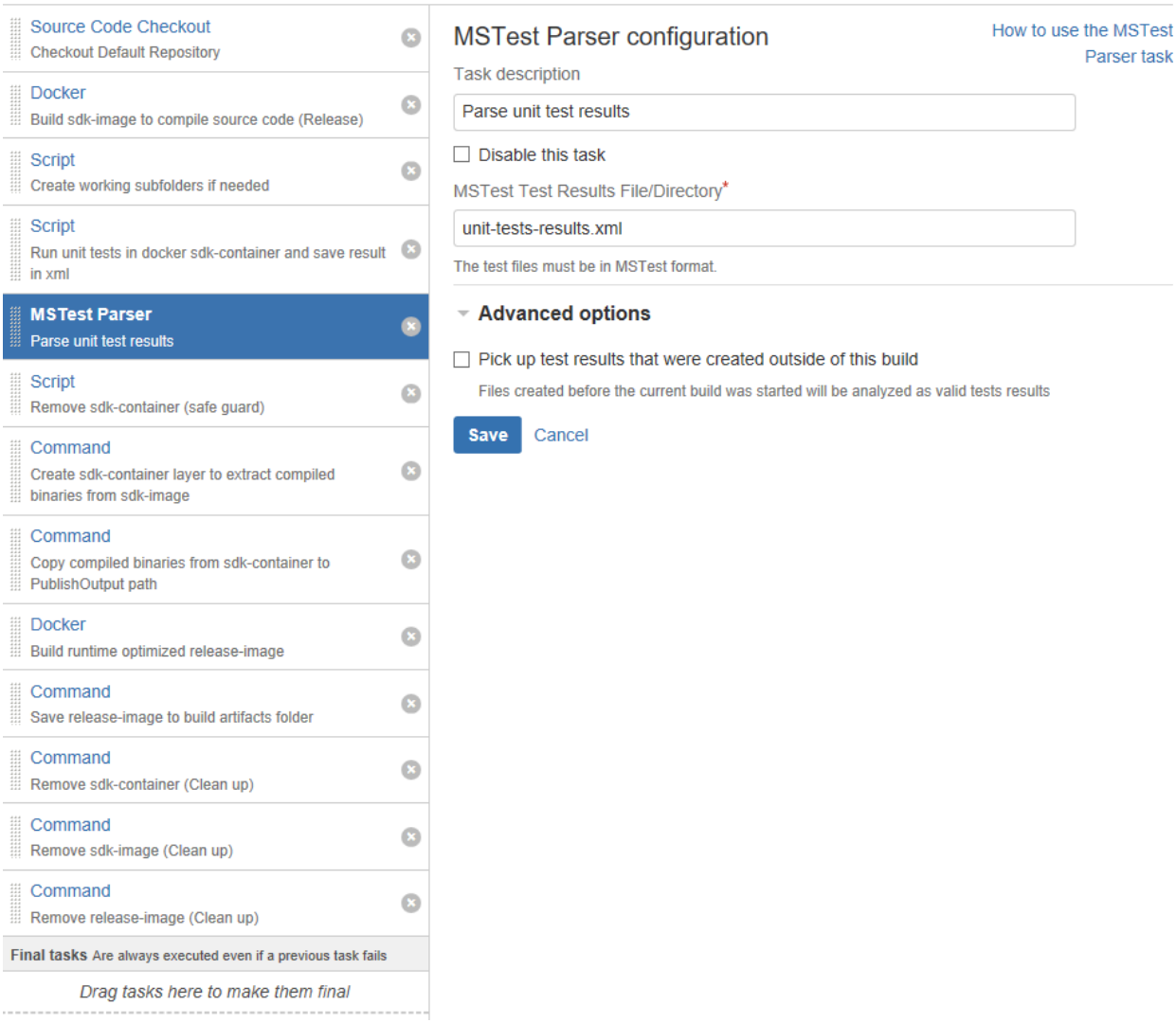
Script body:

```
#!/bin/sh
docker run --volume ${bamboo_build_working_directory}:/data --workdir /app --
rm vcp-netcore-webappdem01:sdkimage dotnet test .\\UnitTests\\UnitTests.csproj
-l 'trx;LogFileName=/data/unit-tests-results.xml'
exit 0
```

NOTE: “exit 0” is used to bypass any error code returned during unit testing. The next step will be the responsible to continue or to stop the CI pipeline execution

Task result: a XML file with the result of the unit tests.

5. Task: “MSTest Parser/ Parse unit test results”

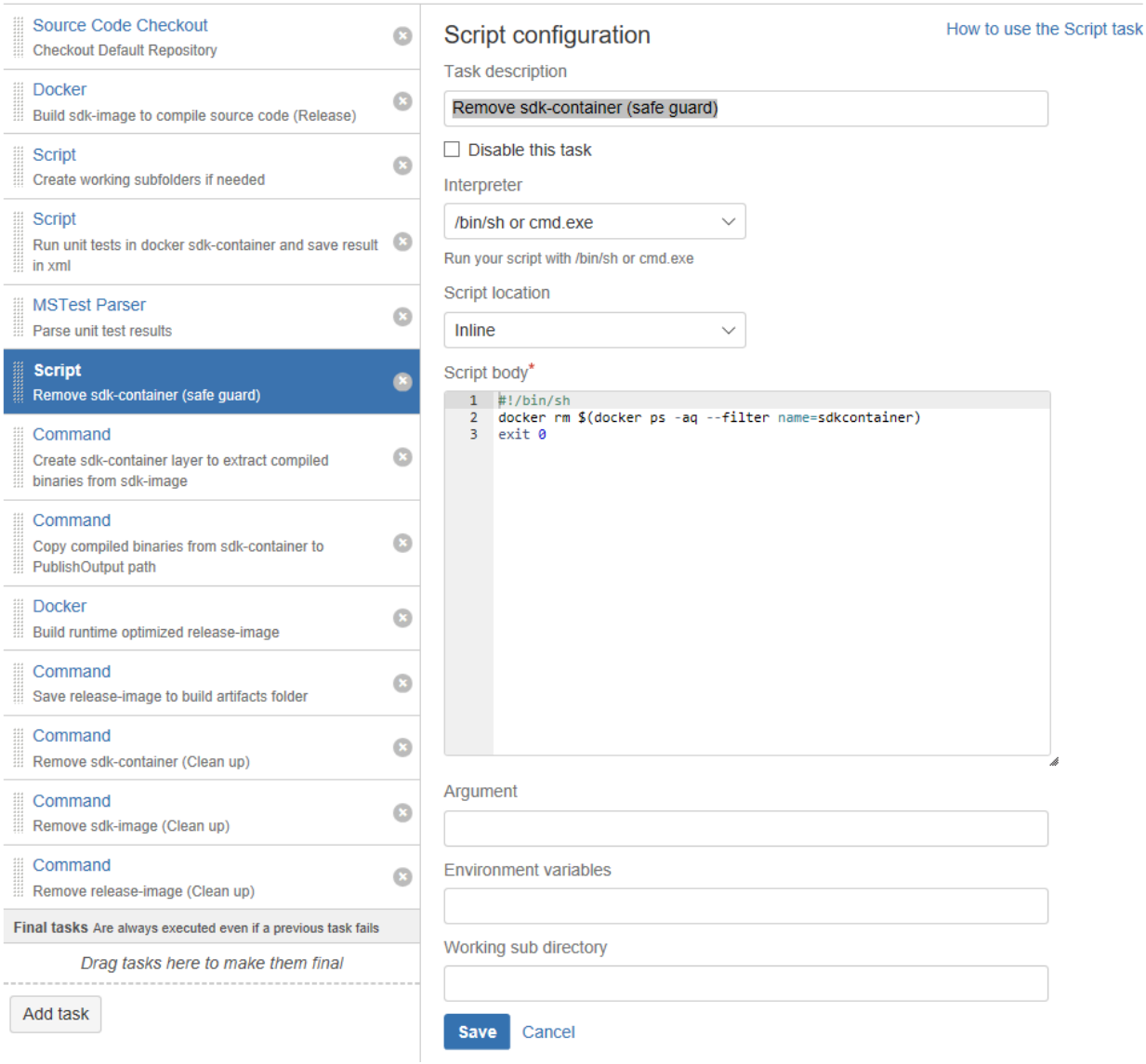


The current bamboo implementation supports the following tests parsers:

- JUnit Parser
- MBUnit Parser
- Mocha Test Parser
- MSTest Parser
- NUnit Parser
- TestNG Parser

Task result: UI report of what happened and a milestone to stop or to continue with the CI execution.

6. Task: “Script / Remove sdk-container (safe guard)”



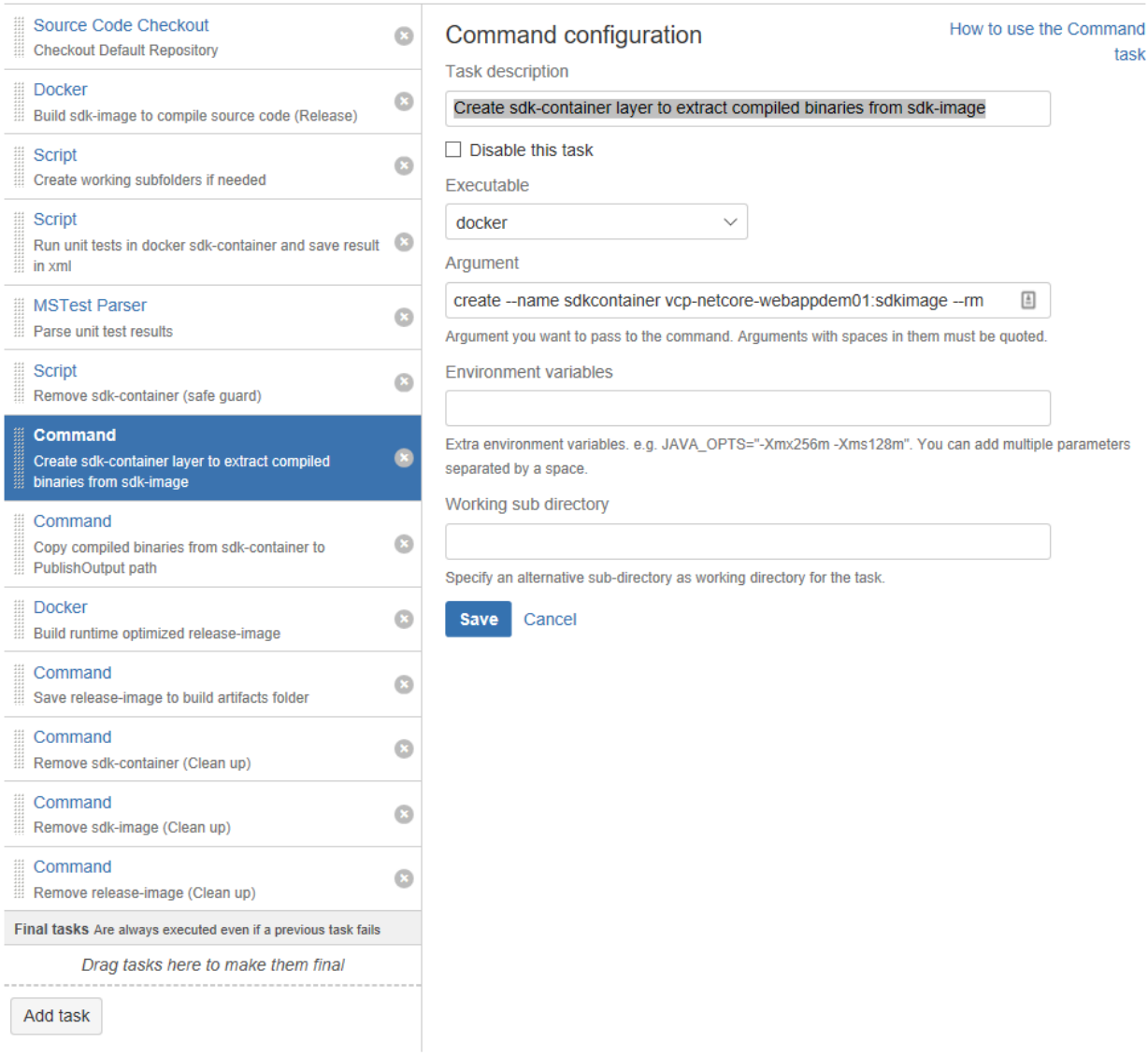
Script body:

```
#!/bin/sh

docker rm $(docker ps -aq --filter name=sdkcontainer)
exit 0
```

Note: This task is a safe-guard to avoid a potential container-name collision in the next step.

Task result: remove a potential container that might be registered in the Bamboo agent that runs this job.

7. Task: “Command / Create sdk-container layer to extract compiled binaries from sdk-image”

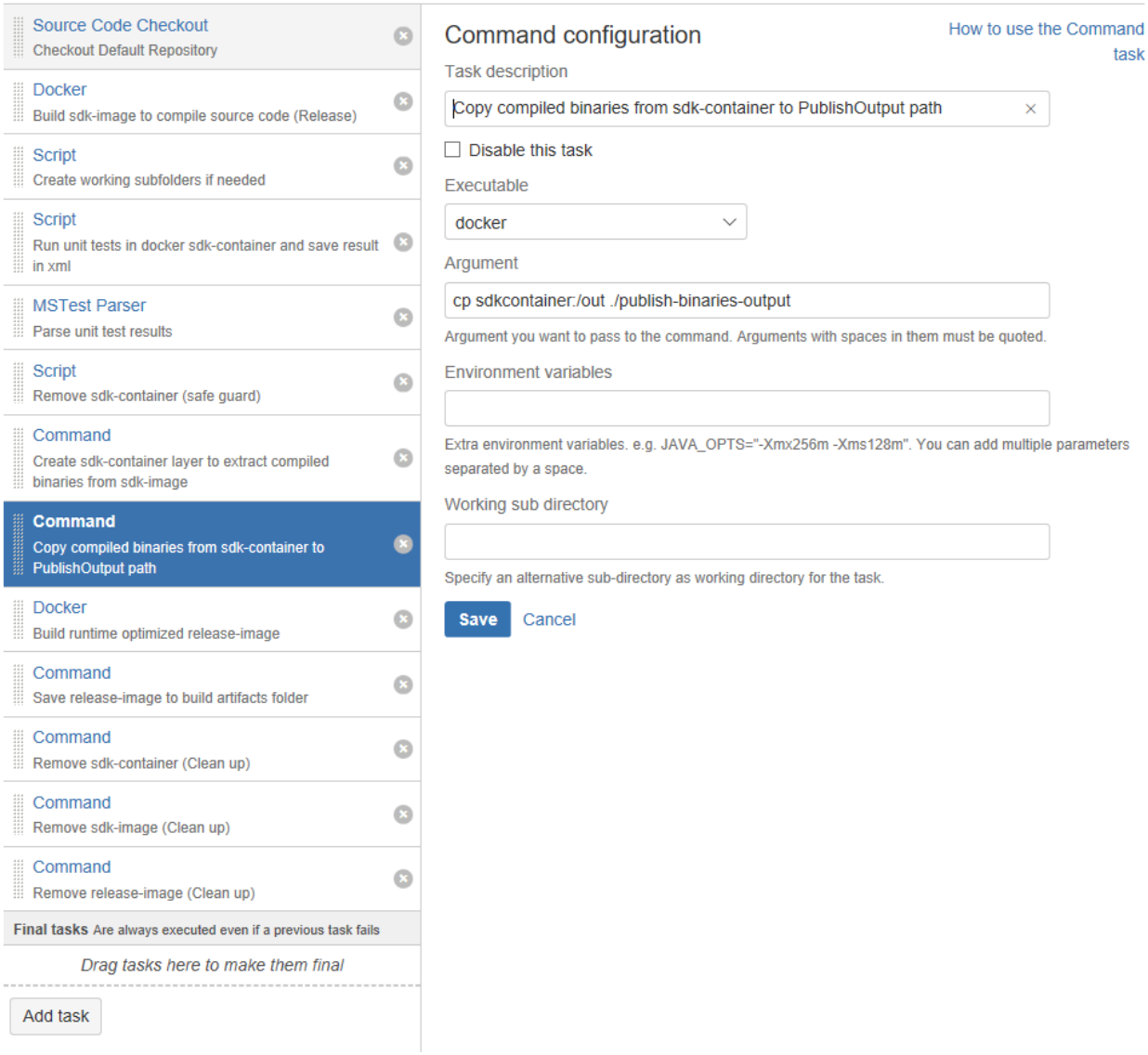


Argument:

```
create --name sdkcontainer vcp-netcore-webappdem01:sdkimage --rm
```

Task result: Creates a container layer in the bamboo agent.

8. “Command / Copy compiled binaries from sdk-container to PublishOutput path”

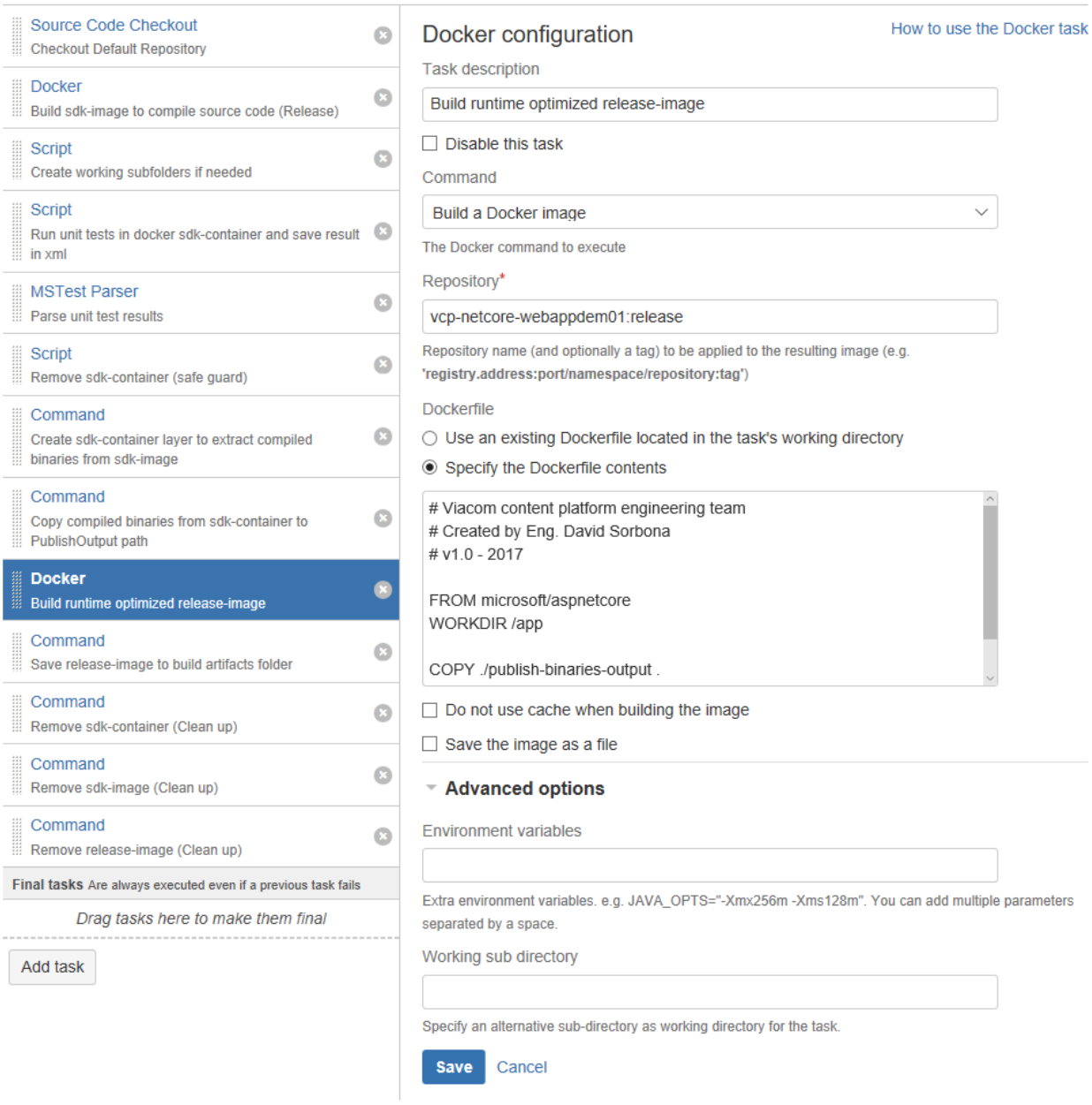


Argument:

```
cp sdkcontainer:/out ./publish-binaries-output
```

Task result: binary files are copied to a bamboo agent folder.

9. “Docker / Build runtime optimized release-image”

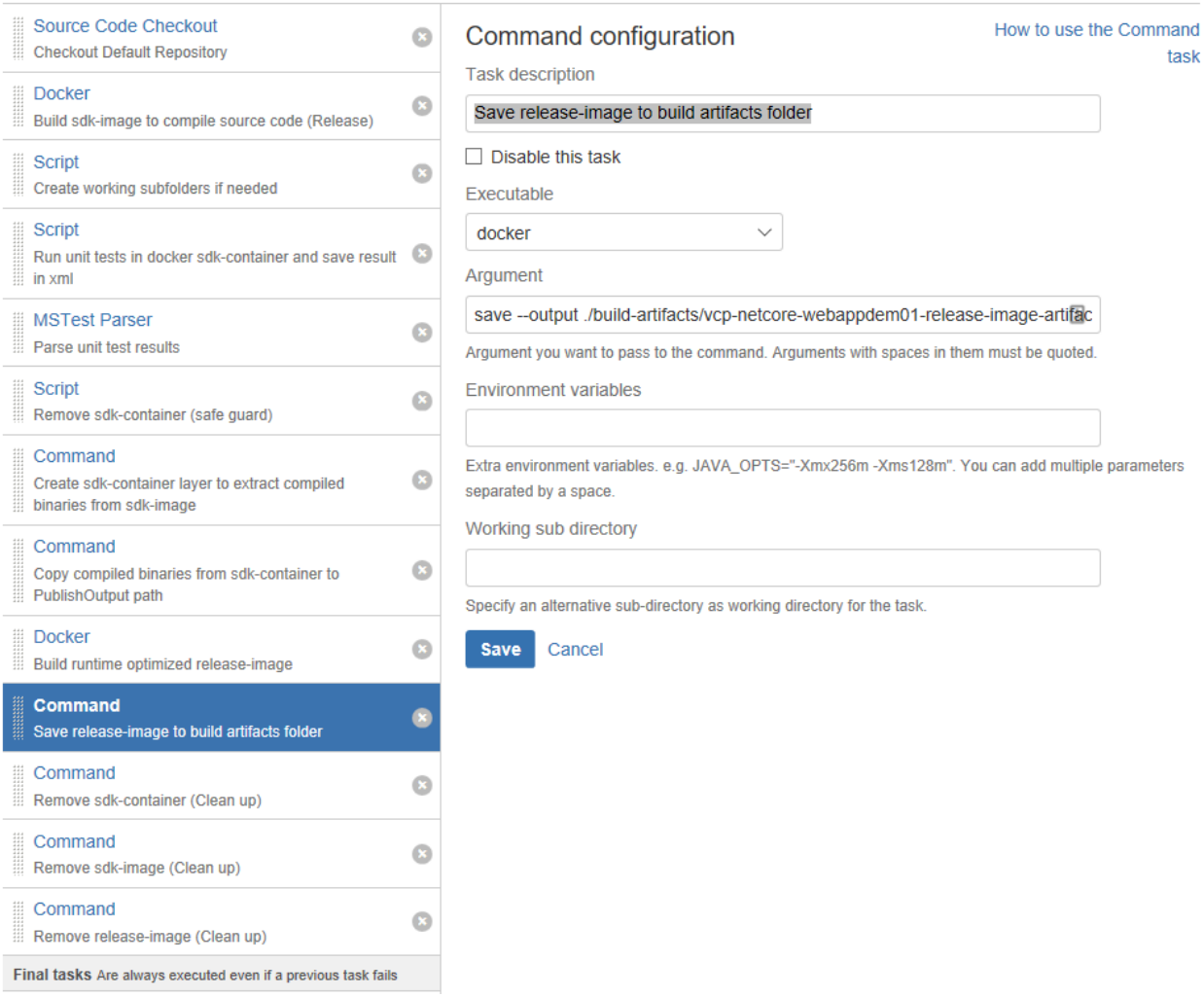


Dockerfile:

```
FROM microsoft/aspnetcore
WORKDIR /app
COPY ./publish-binaries-output .
ENTRYPOINT ["dotnet", "vcp-netcore-webappdem01.dll"]
```

Task result: an optimized Docker image ready to run.

10. “Command / Save release-image to build artifacts folder”



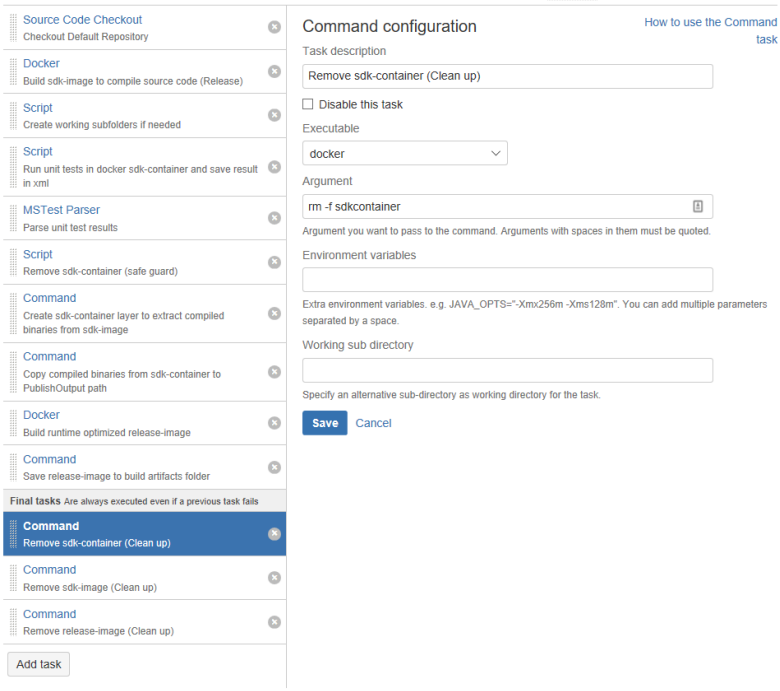
Argument:

```
save  --output  ./build-artifacts/vcp-netcore-webappdem01-release-image-artifact.tar
vcp-netcore-webappdem01:release
```

Task result: The Docker “release-image” packed in a .tar file. This is the artifact produced by this CI/Build pipeline. This artifact is the one that will be used by the “Bamboo deploy project”

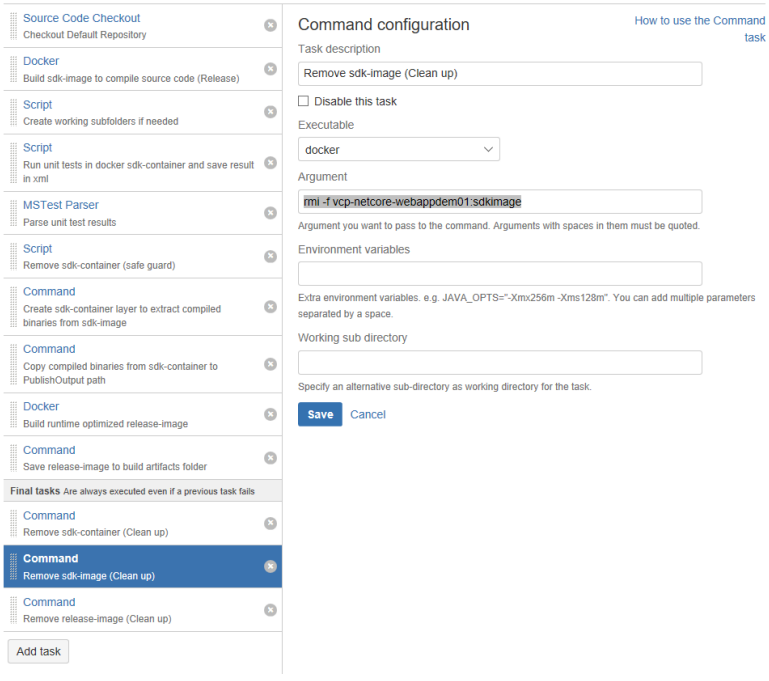
Next steps are used to housekeeping and clean up.

11. “Command / Remove sdk-container (Clean up)”



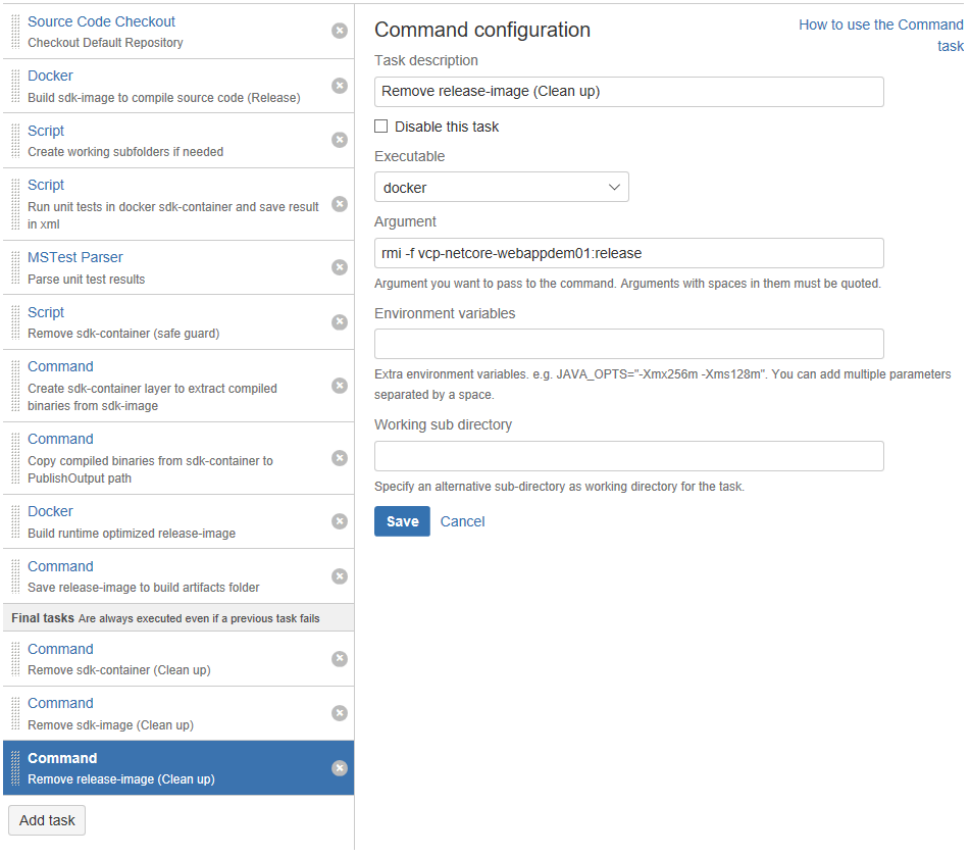
Argument: `rm –f sdkcontainer`

12. “Command / Remove sdk-image (Clean up)”



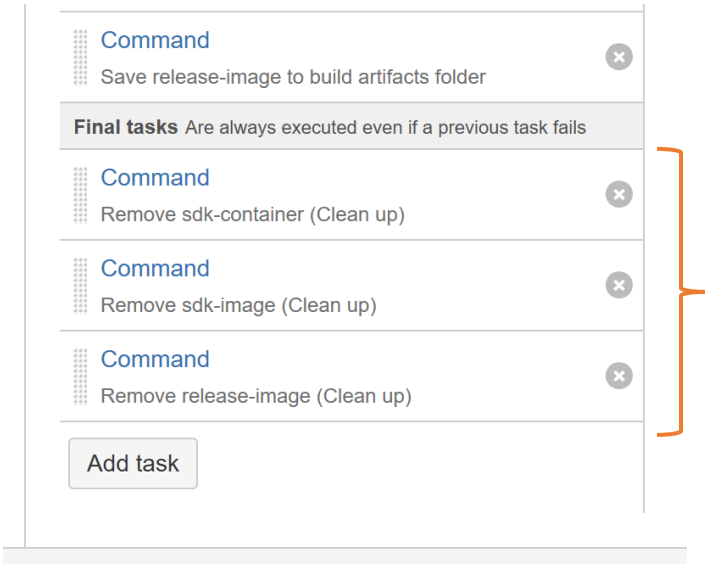
Argument: `rmi –f vcp-netcore-webappdem01:sdkimage`

13. “Command / Remove release-image (Clean up)”



Argument: `rmi -f vcp-netcore-webappdem01:release`

14. Move the last three clean up commands to the “Final task” stage:



15. Done! The CI pipeline is ready.

Why do we use different images for building and for release?

Look at the size column and you will find out:

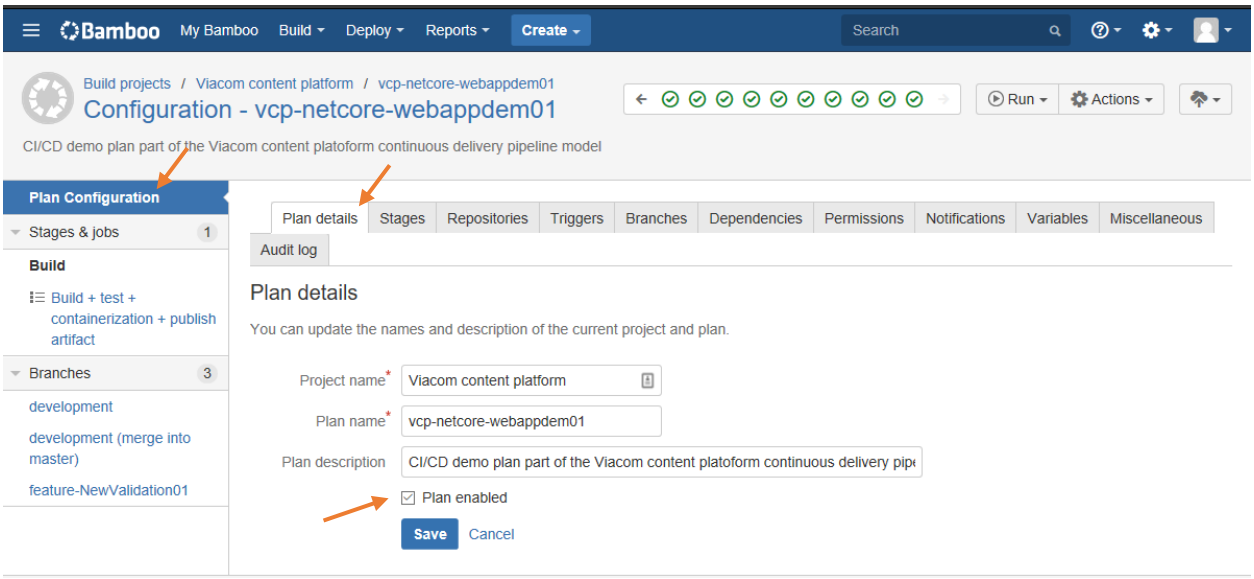
```
PS C:\Users\sorbonad.PLAYASUR> docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED                  SIZE
vcp-netcore-webappdem01     lastest             68d48081c233        About a minute ago       318 MB
vcp-netcore-webappdem01     sdkimage            b54afff3a0cd        6 minutes ago            1.25 GB
```

- sdk-image (tag: sdkimage): 1.25 GB
- release-image (tag: lastest): 318 MB

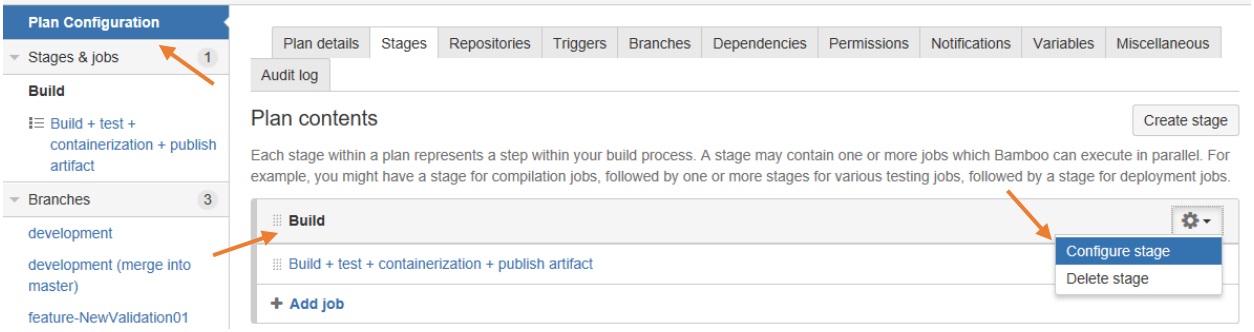
Additional configurations

Plan configuration

1. Make sure your plan is enabled:



2. Rename the stage as “Build”:

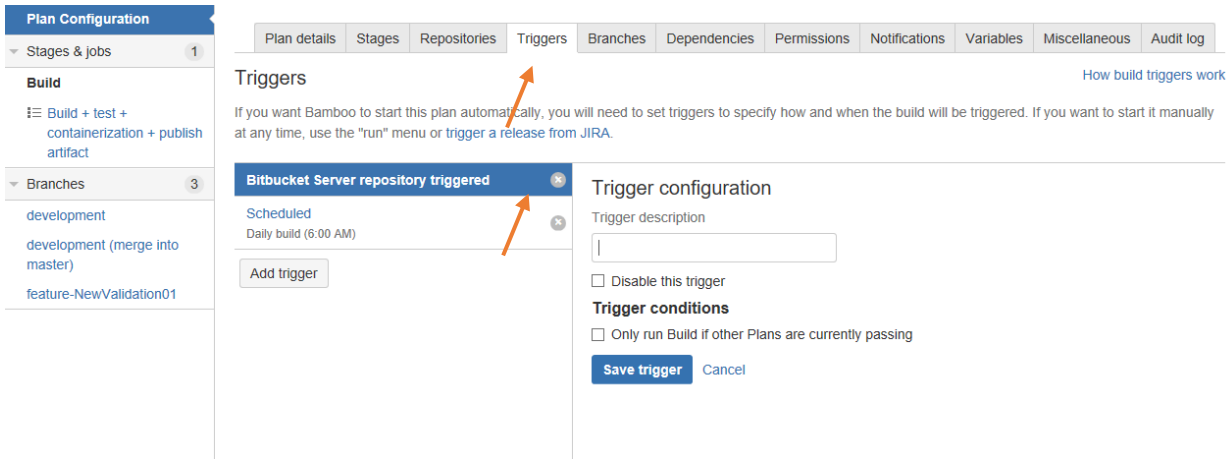


Why are we using only one “stage”?

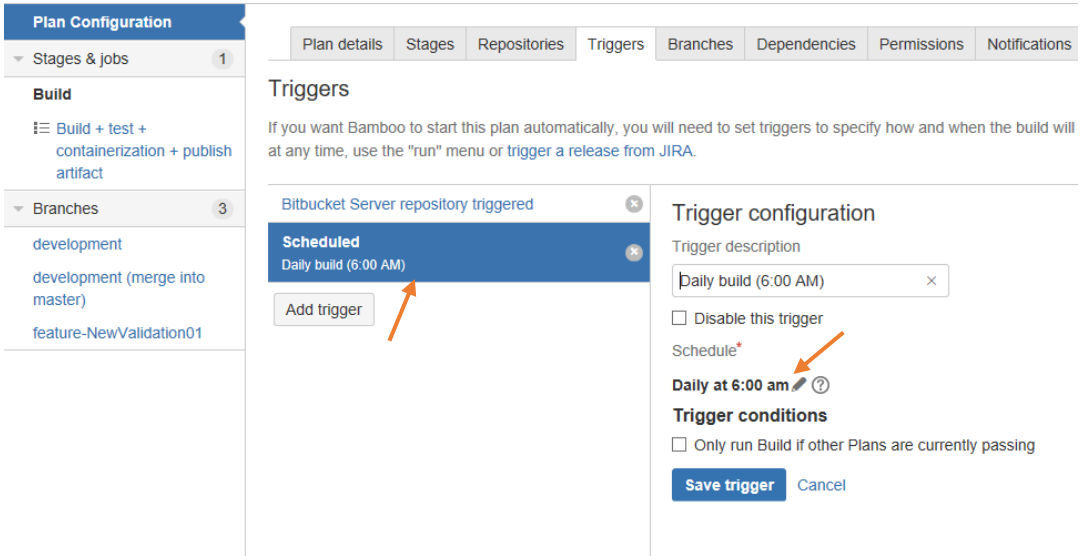
Grouping tasks in stages is a great way to keep complexity low, also every stage runs sequentially so at first glance everything looks good. But during research I’ve found that stages do not always run in the same “bamboo agent”, so eventually one stage could be executed by “Agent A” and the following stage could be executed by “Agent B”; this behavior breaks the “container-approach” of the pipeline. Even though would be possible to pass the container-images between stages it will create a lot of overhead that does not make sense to have and maintain.

3. Setup your “Build plan” triggers:

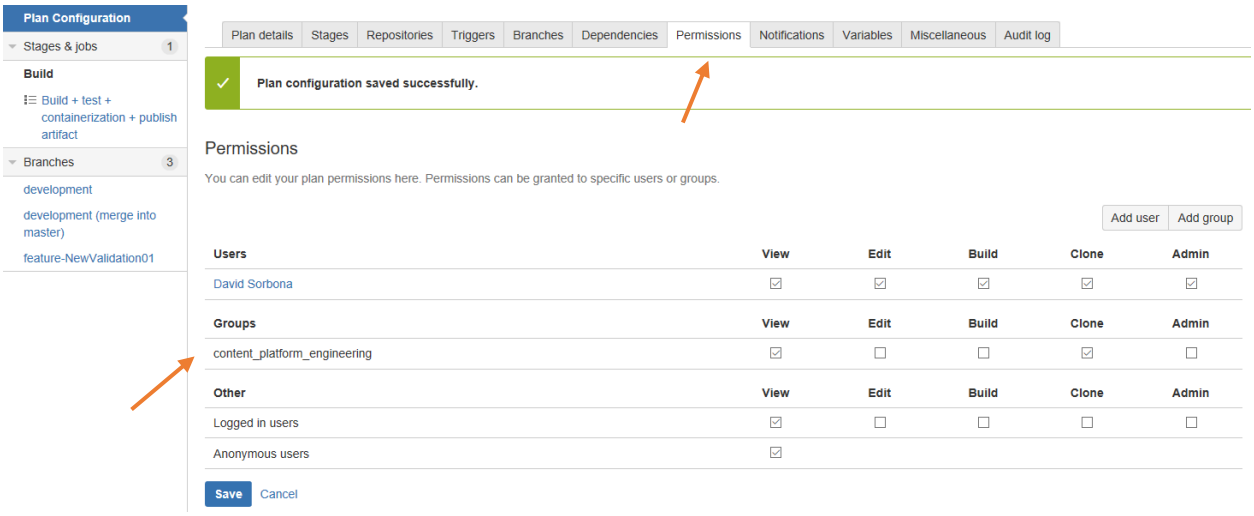
First set a trigger for changes in repository:



Now add a “scheduled” trigger at set it to 6:00 AM



4. Setup the “Permissions”:



IMPORTANT

You could use the Crowd security group “content_platform_engineering” to assign permissions to the whole team.

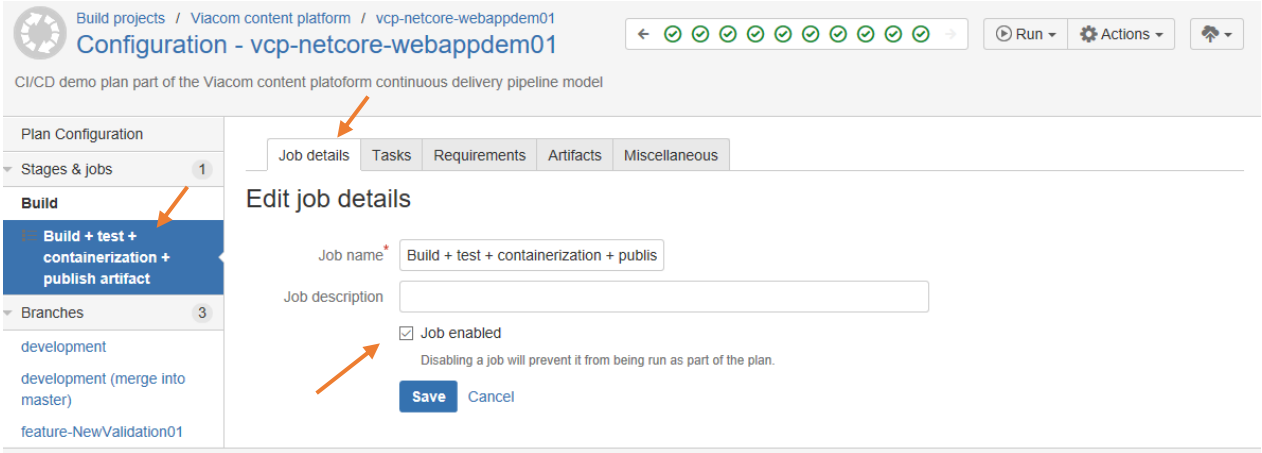
PERMISSIONS

I suggest this schema of permissions:

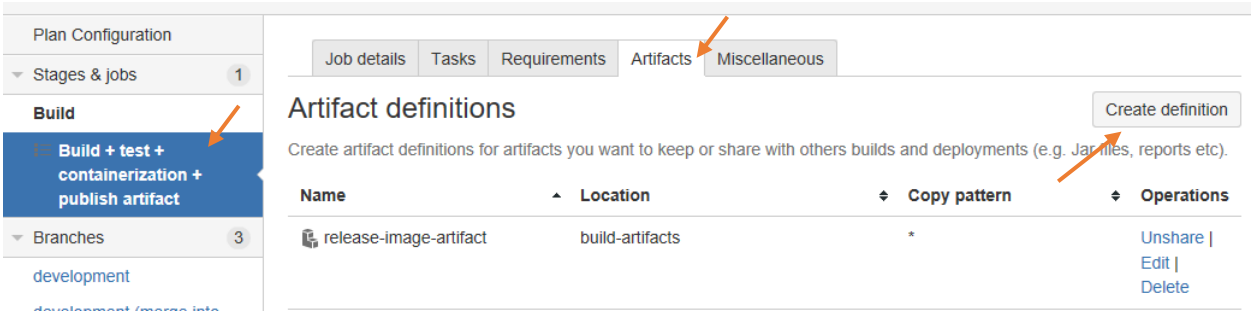
- Architects/Product owners -> admin.
- Application principal engineering -> admin.
- Rest of the team -> View.

Job configuration

1. Make sure your job is enabled and rename it as “Build + test + containerization + publish artifact”



2. Create an “artifact definition”



Then complete the form with these values:

NOTE

This configuration defines the location where the artifact will be copied and the copy pattern used to copy the files during the deploy step.

Branch management

Up to now the default job of the “build plan” is using the code from the “master” branch. Bamboo allows us to change the branch after the “Checkout” task based on “Branch plans”, thus you can execute the same building pipeline (same task, same job, same stages) using different code bases, cool right?

Git-flow predefines a set of branches that has particular responsibilities, bamboo allows you to automate the “merging” of those branches automatically.

IMPORTANT: In this version of the procedure we are using a subset of the branches defined by a git-flow since more research and deeper analysis is needed. I do this to avoid a potential “hell branch” scenario and to keep the development workflow experience as simple as possible. It could change in the future based on the feedback received from developers.

Setup branches

As was defined before, we have four kind of branches: “master” (created by default during repo creation), “development”, “feature” and “personal”.

If you are creating a new repository from scratch you have to create the “development branch” thru git, for doing that follow the next steps:

1. Open a command line and go to the working directory of your repository.

```
cd .\vimn-america_code-repository\Viacom.ContentPlatform\vcp-netcore-webappdem01\
```

2. To create the development branch in your local repo and origin (remote) repo, type this:

```
git branch development

git checkout development

git push --set-upstream origin development
```

3. Done! You have the development branch ready

WARNING: You do not code in the “development branch”, it is only an integration branch use by bamboo and will be only one “development branch”.

If you want start coding, you have to create a feature branch. First define its name with this naming conventions:

```
feature/[Name of the feature of group of features]
```

"feature/" is mandatory and key sensitive, so watch out. The right side of the name is up to you; it should be a name that represent "**the feature o group of features you want to integration into the development branch**". Here some examples:

feature/NewValidationForOrderRequest

feature/JIRA-ISSUE-KEY (this way you will link the Jira ticket with the branch plan)

feature/CumulativeUpdate-May2017-week1

feature/Refactory-May2017-week1

feature/CumulativeUpdate-Sprint11

Having a good name is the most difficult part, then you just have to run these commands:

```
git branch feature/NewValidation01

git checkout feature/NewValidation01

git push --set-upstream origin feature/NewValidation01
```

Finally, you can check if all branches are there by running this command:

```
git branch -a
```

Result:

```
PS C:\vimn-america_code-repository\Viacom.ContentPlatform\vcp-netcore-webappdem01> git branch -a
* development
  feature/NewValidation01
  master
  remotes/origin/development
  remotes/origin/feature/NewValidation01
  remotes/origin/master
PS C:\vimn-america_code-repository\Viacom.ContentPlatform\vcp-netcore-webappdem01>
```

You should see 3 local branches and 3 remote branches, if not then something went wrong. Do not continue until the branch list looks like that.

IMPORTANT needs review

Speaking in general, once a "feature/" is complete, all the tasks completed and you have confirmation that your code was integrated into the "development branch" and all unit tests passed, then the "feature/" branch should be disabled or deleted.

Eventually there could co-exists multiple "feature/" branches, in that case be careful about potential "integration merging conflicts". This scenario could exist when multiple developers work on the same code base with multiple different "features/" branches or multiple developers work in the same "feature/" branch. In any case, you should avoid that the "development branch" and the "feature/" branch starts diverging too much, otherwise the possibility of "merging conflicts" increases. If that happens then you should explore the "git rebase" command before pushing your code to bitbucket.

https://git-scm.com/book/en/v2/Git-Branching-Rebasing

Rules for working with branches

- If another developer is working in the same feature branch: **pull often.**
- If another developer is working in another feature branch and you work took longer that you thought: **rebase your code.**
- If you break integration step or unit tests**, you fix it.**

Decision point – Option A

If you use pull request strategy to integrate code from "feature/" to "development" you can jump to the section: Setup predefined plan branch for "development branch".

Decision point – Option B

If you are using automatic "feature/" to "development" integration strategy, then you have to create a "personal/" branch from the feature branch. You can do it like this:

```
git checkout feature/NewValidation01

git branch sorbonad/feature/NewValidation01

git checkout sorbonad/feature/NewValidation01

git push --set-upstream origin sorbonad/feature/NewValidation01
```

Setup branch plans for automatic integration strategy

The "Bamboo branch plans" allows you to automate code integration (marge between branches) and create new plans on the fly based on branch name conventions.

The general idea is that you create your feature branch and push code to bitbucket, then bamboo will detect the new "feature/" branch and automatically will create a "branch plan" that will integrate the code from that "feature/" branch into the "development" branch. After that, the "release management lead" or "QA lead" can manually execute another plan that will integrate the code from "development branch" into the "master" branch.

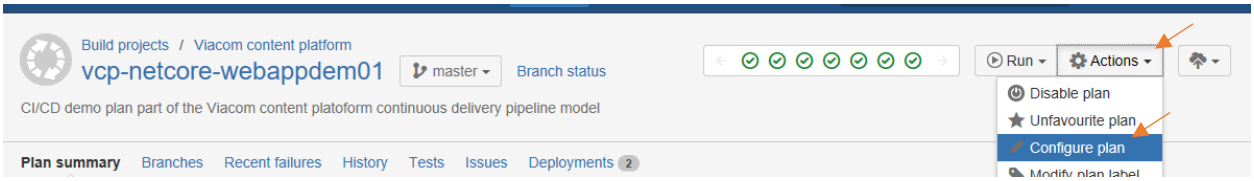
WARNNING

Only the "Release manager" or "Tech lead" can integrate code into master and only by executing manually a predefined "branch plan" in Bamboo.

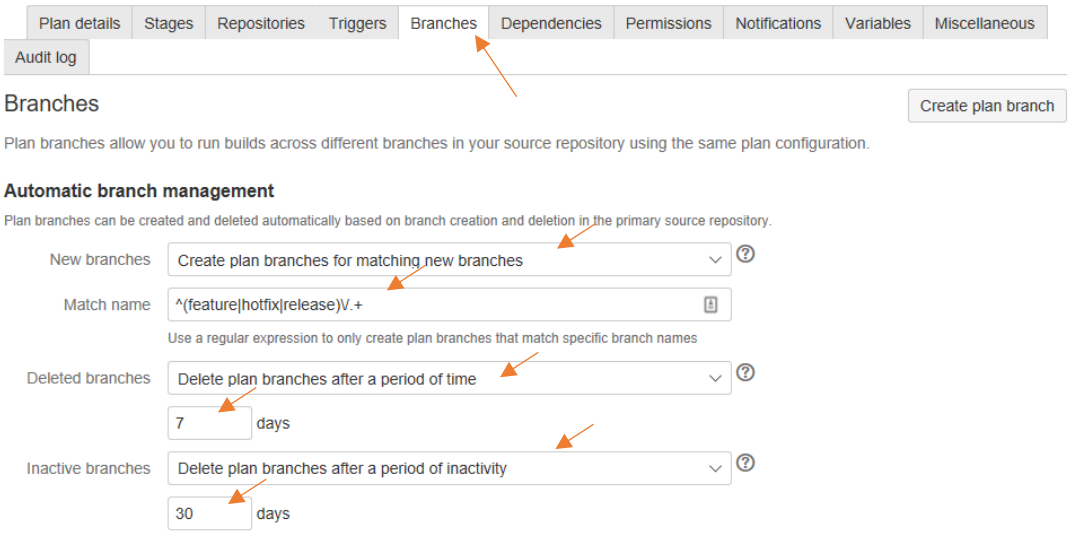
Setup automatic plan branch generation for feature branches

To setup the automatic plan change follows the next steps:

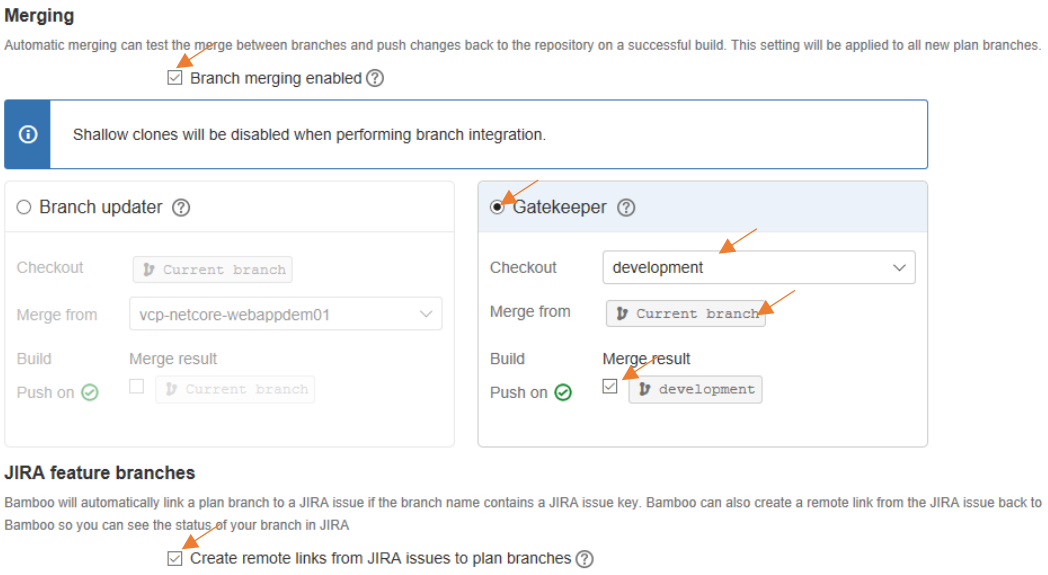
1. Go to the Build plan configuration portal

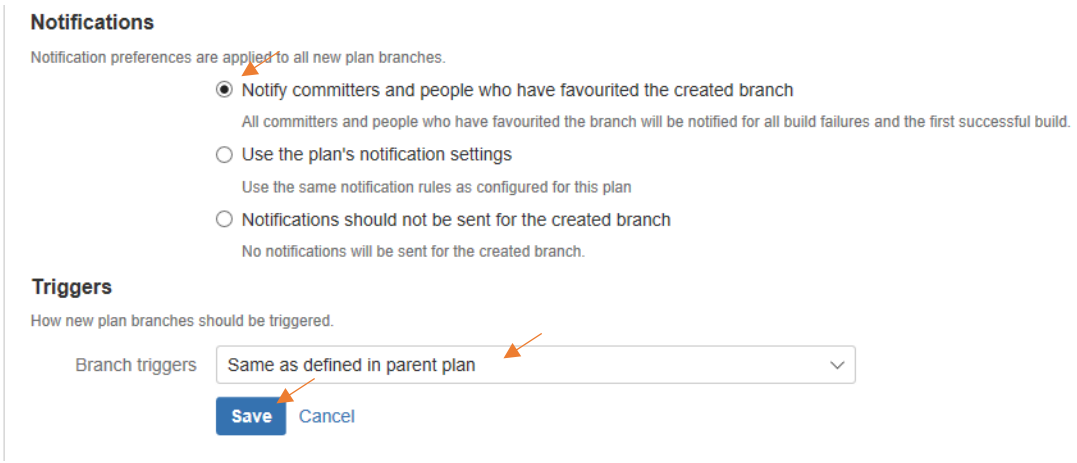


2. Go to “Branch” and fills the form as is shown below and save it:



Match name regular expression: `^(feature|hotfix|release)\/.+`



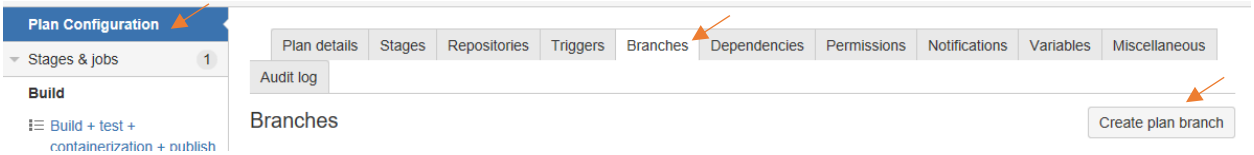


3. Done! You have the automatic integration plan setup. From now for any new “feature/” branch created in the main repository in bitbucket, bamboo will create a “plan branch” that will trigger the CI pipeline using the merged code between development and the feature/ branch and will integrate the changes into the development branch if nothing fails.

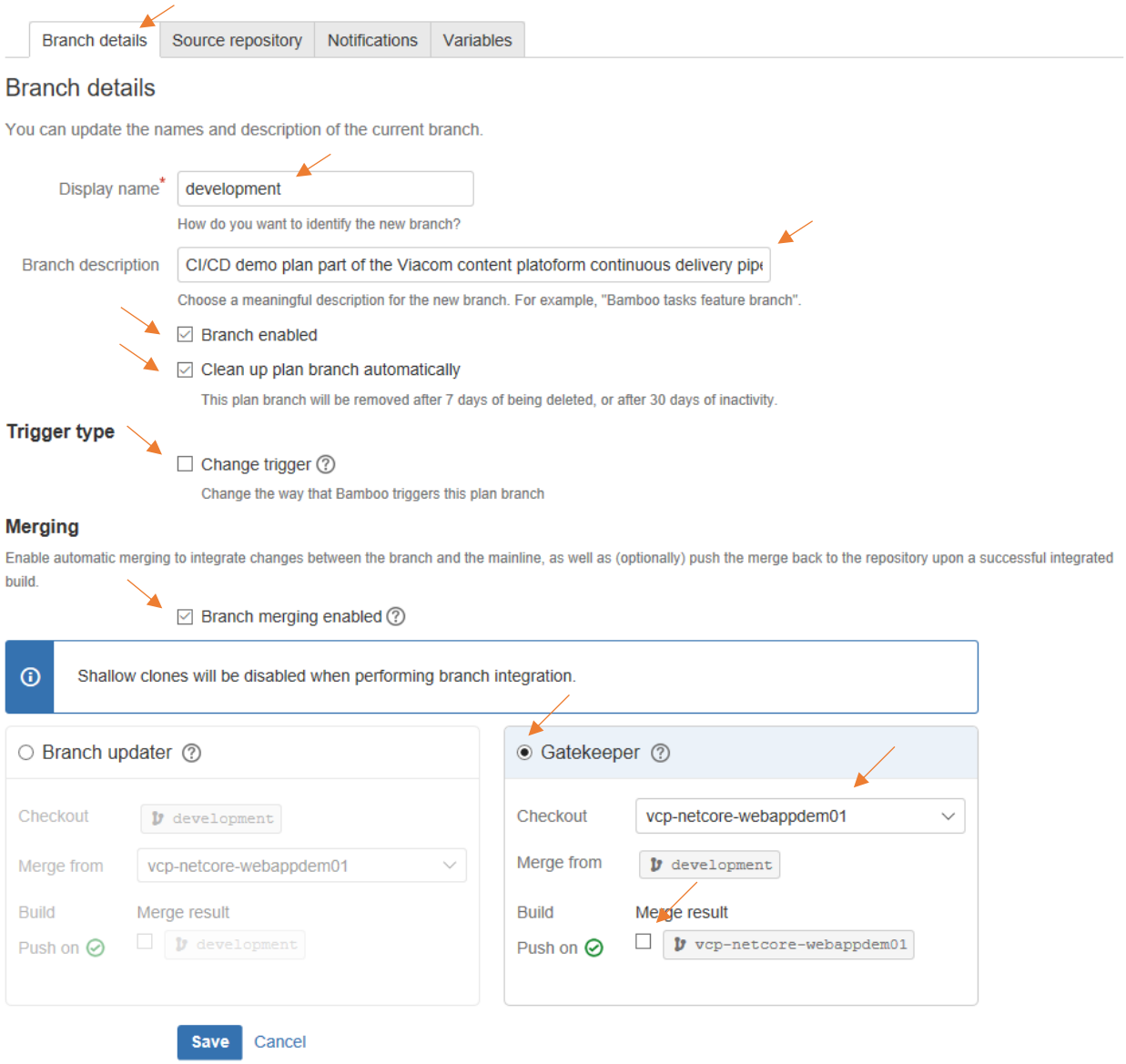
Setup predefined plan branch for “development branch”

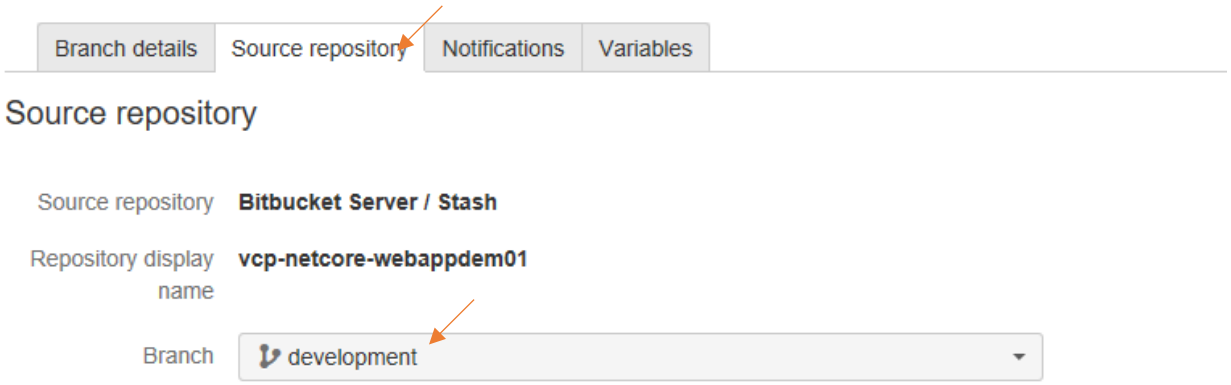
Now let’s create a predefined branch plan for building and testing the “development branch” when a changeset is introduced in the bitbucket repository.

1. Go to the branch tab of the build plan and click on “Create plan branch”



2. Completes the form as shown and save it





3. Save it and done! From now when a changeset is introduced into the code repository (development branch) hosted in bitbucket, bamboo will trigger the CI pipeline (Continuous integration). This allow you to understand if you code can be complied, all unit tests passed and the release artifact can be created.

IMPORTANT

Actually, you can live without this branch plan, but I strongly suggest to have it in place so you can detect any issue in your code as soon as possible. If you do not integrate/test/build often you could detect a killer-issue too late.

NOTE: After executing the instructions explained later in this document, this pipeline will also trigger the CD (continuous deployment) for DEV environment too. That will close the loop for the development environment.

Setup predefined plan branch for "development branch with master integration"

This plan branch is the key of the whole workflow. This is the way you tell the system: "OK Bamboo, I want that the code I've been working on goes to production". In real life, before doing this you executed functional tests in the development environment, talked with users and make sure that the system will work.

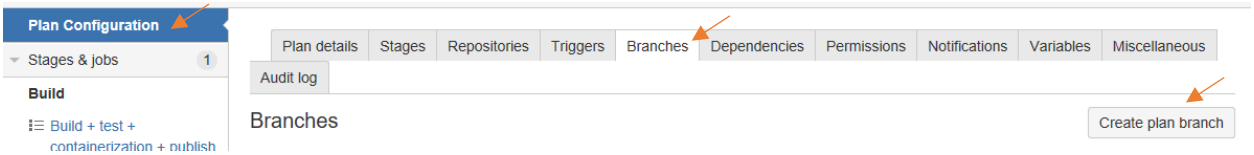
I will call the fact of sending code to production as "**flights**", so by running this plan branch your setup a "flight to production". Sometimes a flight could be delayed for some reason, it means the master branch is ready but the artifact wasn't deployed to the execution environment (for example PROD AWS ECS). I will call that scenario as "**delayed flight**". If a flight gets cancelled ("**cancelled flight**") put in contact with the engineering team board right away.

flight	code that is flying to production.
delayed flight	code that is flying to production (mater branch is updated) but for some reason you have paused the trip.
cancelled flight	code that was going to production but you cancelled for an unexpected situation.

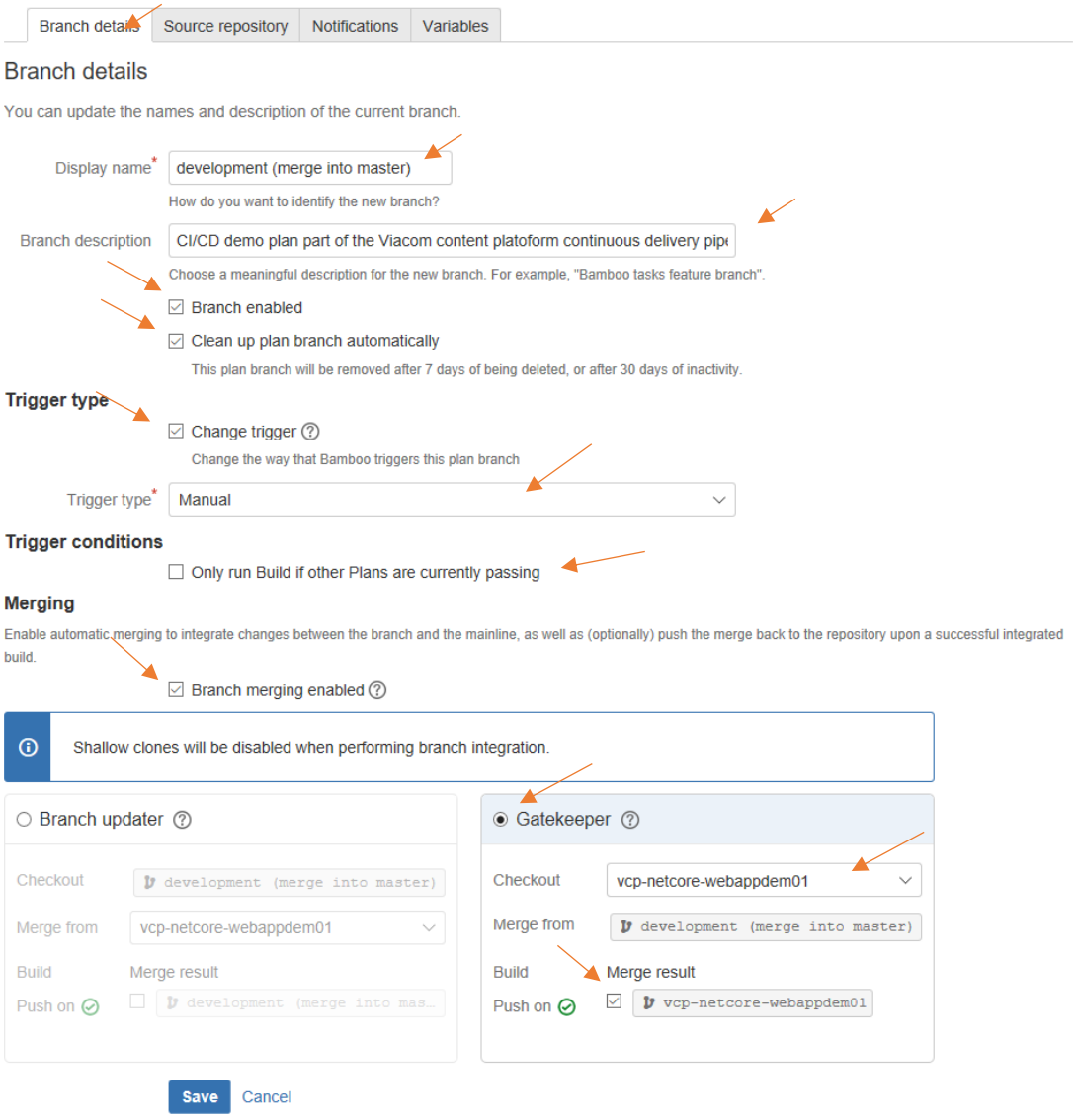
REMEMBER: The master branch hosts the code that is in production or that will be in production pretty soon. (pretty soon = in 1 or 2 days)

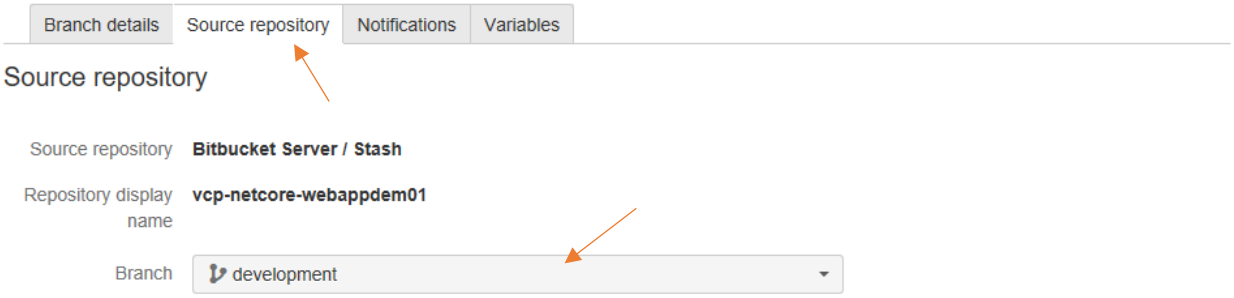
To setup this branch plan follows the next steps:

1. Go to the branch tab of the build plan and click on “Create plan branch”



2. Completes the form as shown and save it





3. Save it and done! This plan branch will allow you to integrate the development branch into the master branch and create a flight to production.

NOTE: After executing the instructions explained later in this document, this pipeline will also trigger the CD (continuous deployment) for DEV environment too for final verification, then you manually will be able to send the flight to QA or PROD execution environments.

Additional observations for non-executable artifacts.

If you project is a non-executable code library that eventually would be used by other projects (shared components), the process is the same but instead of sending the flight to an execution environment you will send it to a repository or component registry, for example to a npm package registry/repo or nugget.

Container registry

Introduction

The container registry is just a place somewhere where you store container images, this procedure defines it as the “Viacom private container repository”.

The registry could be created in several ways, for now we are focusing in the registry service provided by the Viacom AWS instance thru its PaaS “EC2 Registry Container” (AWS ECR)

Pre-requirements

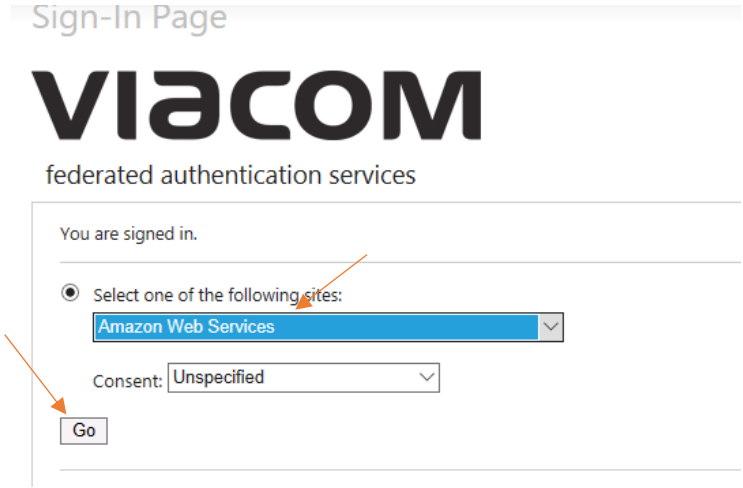
In order to do this, you need access to the Viacom AWS instance. The access is managed by the MCS team (Multiplatform cloud services)

AWS ECR setup

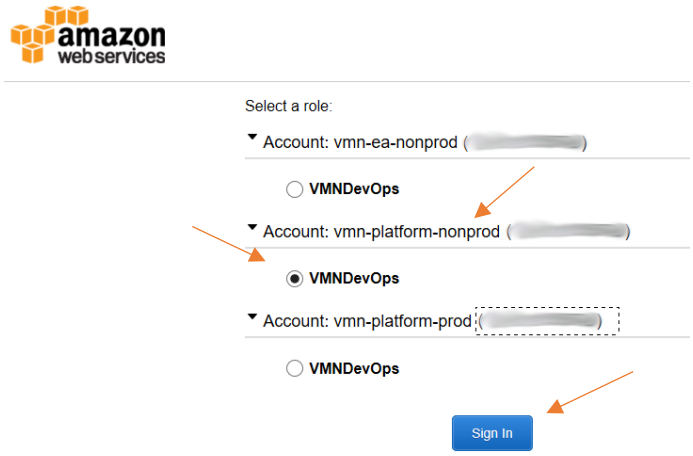
The follow steps show how to create the repository in the Viacom non-prod AWS environment:

1. Log into AWS thru SSO page using your AD credentials.

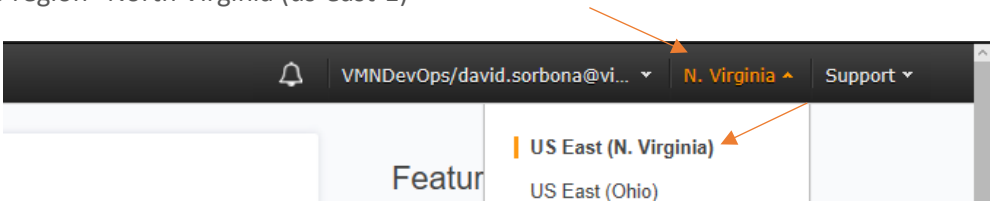
   https://fs.viacomcloud.com/adfs/ls/IdpInitiatedSignOn.aspx



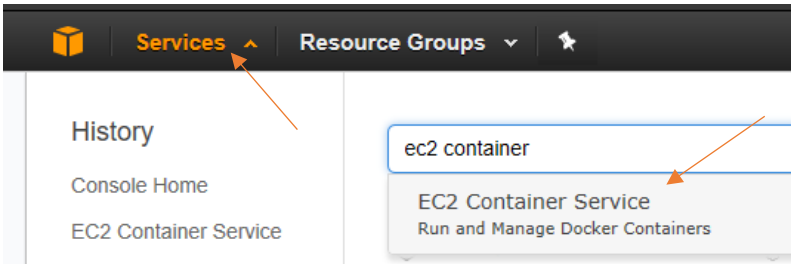
2. Select the account “vmn-platform-nonprod” and sign in.



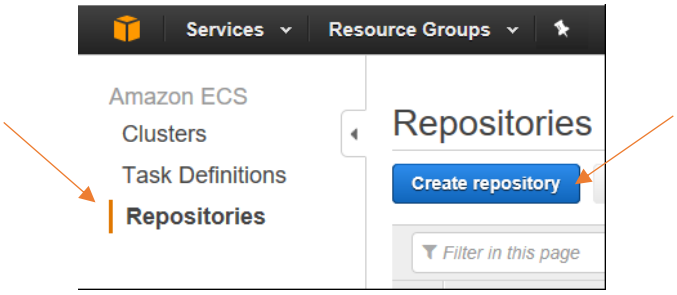
3. Go to region “North Virginia (us-east-1)”



4. Go to “EC2 Container services”



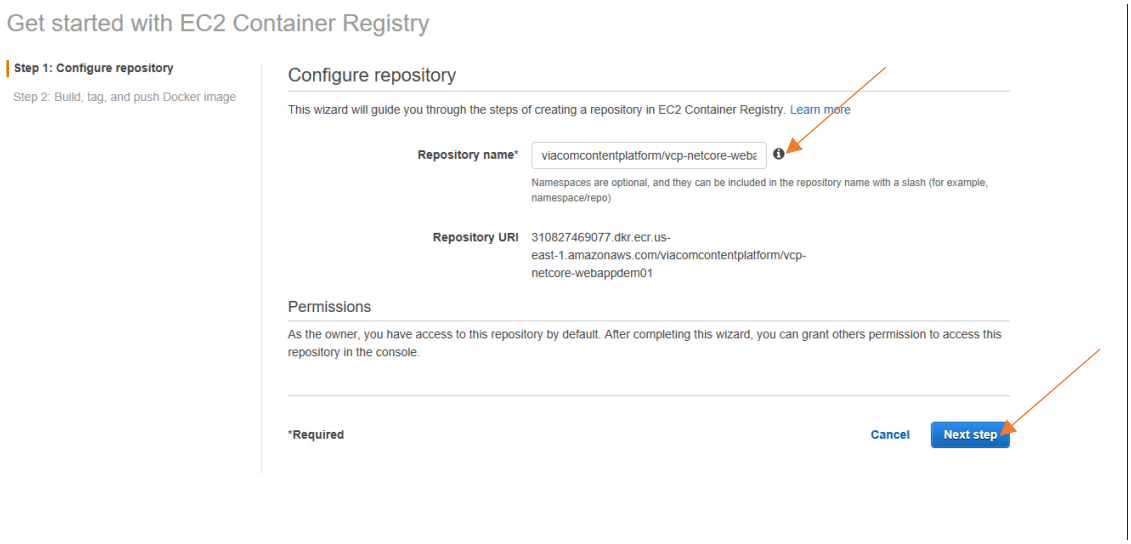
5. Go to “Repositories” and create a new one:



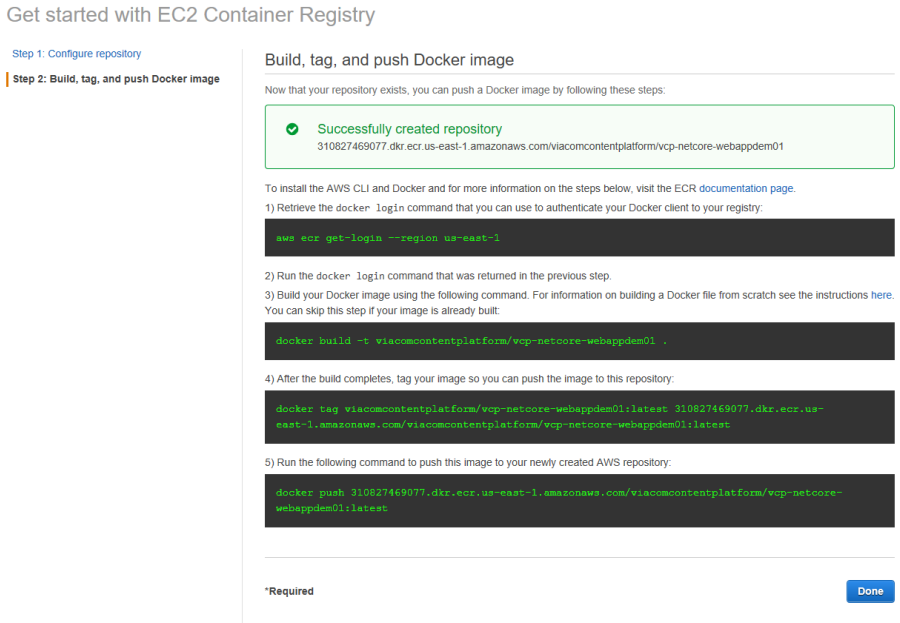
6. Set the repository name and click on go to next step:

   Naming convention : **viacomcontentplatform**/[code repository name]

   for this example, the name is: **viacomcontentplatform/**vcp-netcore-webappdem01

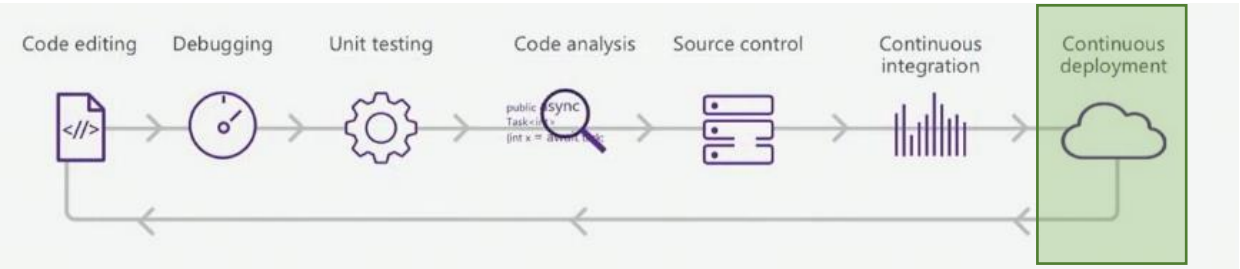


7. Done! You've created a container-image repository on AWS ECR. The last page of the wizard will give you the "aws command line client" instructions to push the container-images. This information will be used for creating the "deploy projects" in Bamboo, so keep it handy.



Continuous deployment pipeline

Introduction



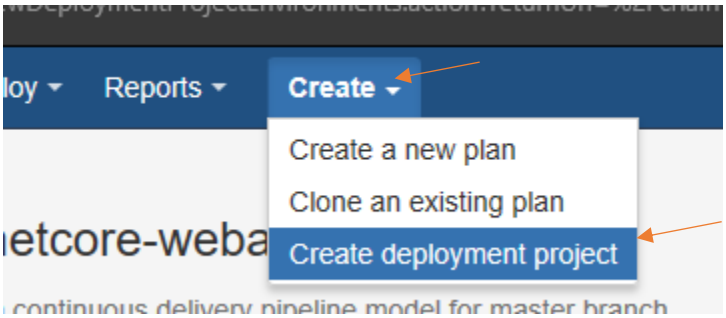
In the previous sections, we saw how to create an artifact (the “release-image” container) that will be deployed in some execution environment (AWS). This section will show how to publish that artifact to a container image repository by creating “deploy projects” in Bamboo.

Pre-requirements

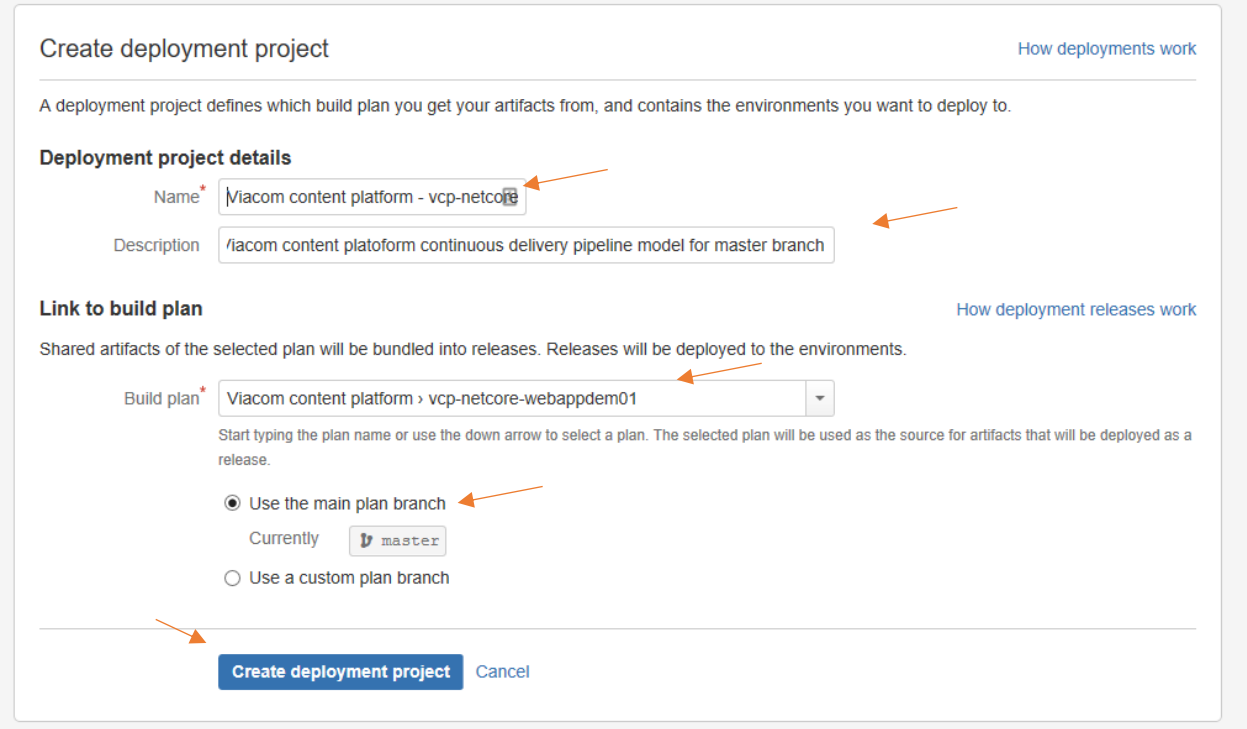
To continue you had to have created a container registry, if not please jump to the section “Container registry” to set it up and then come back to setup the “deployment project”.

Deployment project setup for “master branch”

1. Create a new “deployment project”



2. Fill the form



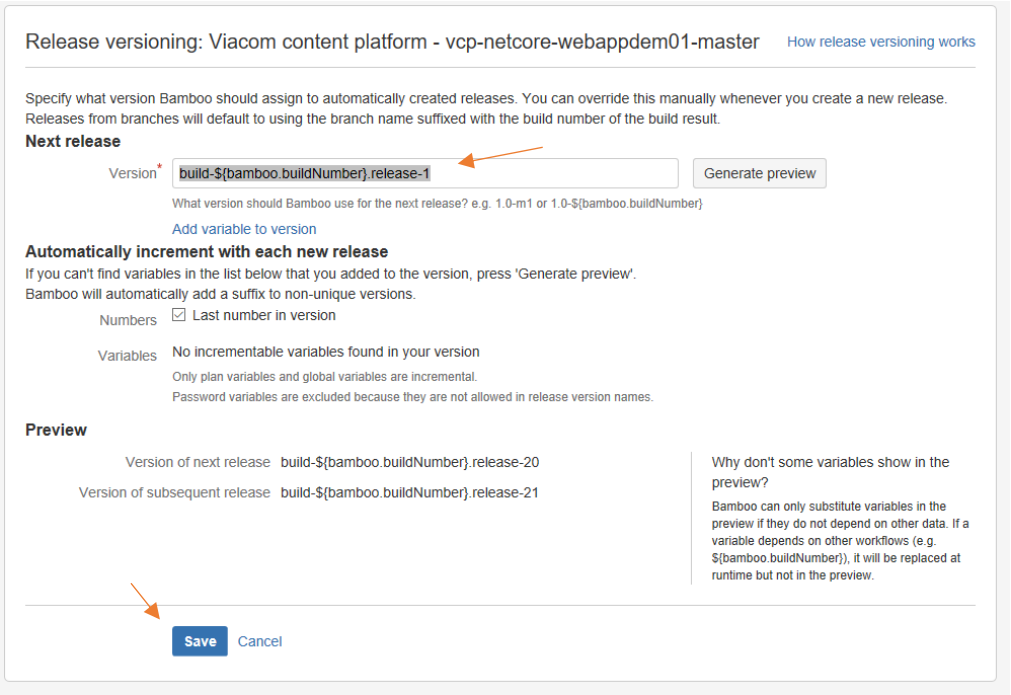
Naming convention:

```
Viacom content platform – [repository name]-[branch name]
```

In this example:

```
“Viacom content platform - vcp-netcore-webappdem01-master”
```

3. Set release versioning



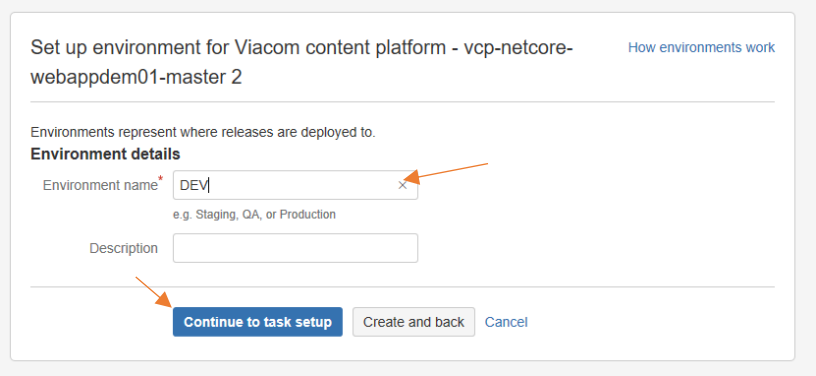
Version:

```
build-${bamboo.buildNumber}.release-1
```

This way we will have track of build number and release number.

NOTE: Besides this automatic versioning I strongly suggest using "semantic versioning" (http://semver.org/) to identify releases during deployment, e.g.:"v1.2.3-build-532.release-23" PENIDNG TO TEST

4. Add environments: DEV, QA and PROD.



The "Environment" is how Bamboo represents an execution environment. Set the name and click "Continue to task setup"

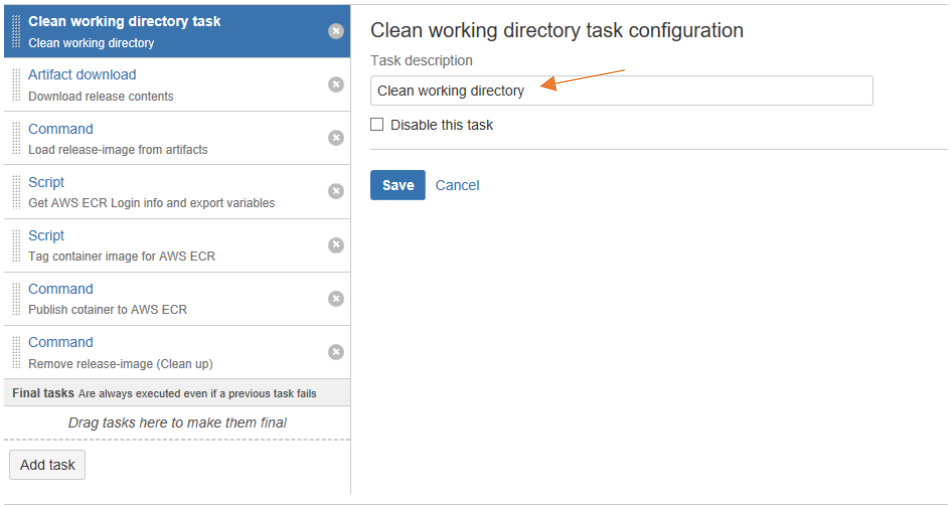
NOTE: The QA and PROD environment will be created later by cloning the DEV environment.

Deployment project tasks setup

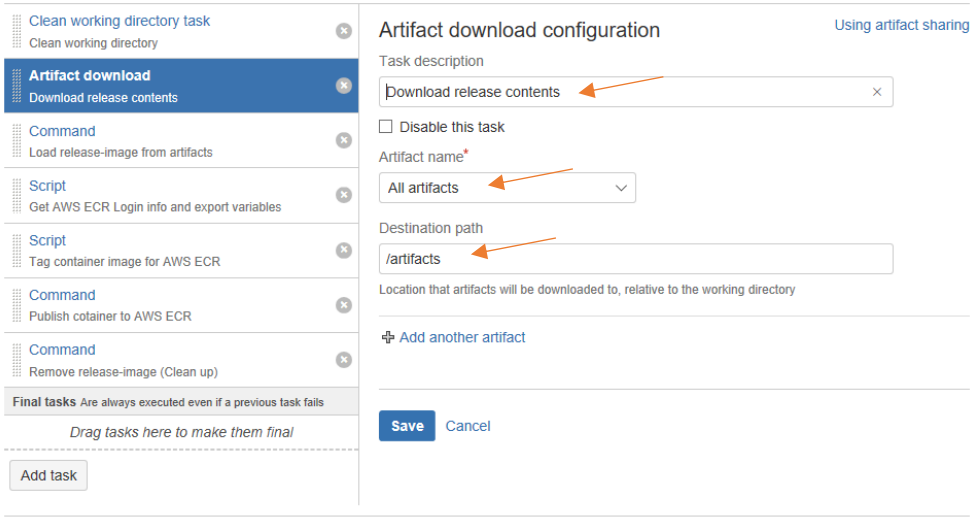
In Bamboo for every "environment" we will have a sequence of tasks that will be executed during the deploy. In a nutshell, we are going retrieve the "release artifact" generated previously by the "building plan", get credential for AWS and pushing the "artifact", as a container image, to the AWS ERC. In the last step we will command AWS to update the "Service" by sending a new "Task definition" to the "Viacom Jenkins based cloud orchestration and automation tool".

For this example, the tasks to be created are:

1. "Cleaning working directory task configuration / Clean working directory"

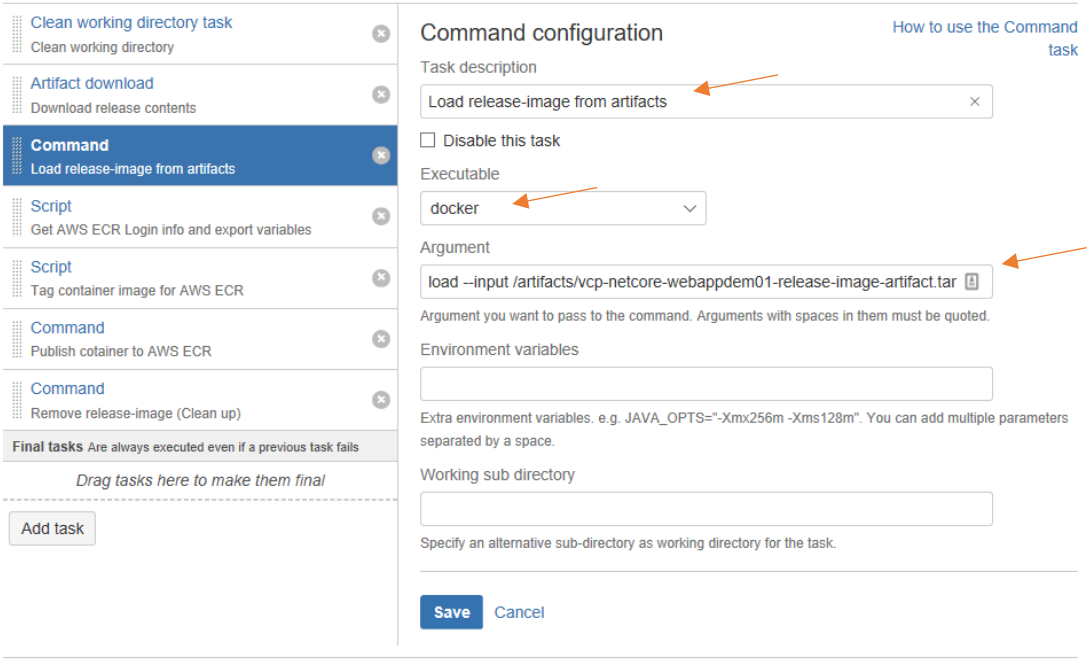


2. "Artifact download / Download release contents"



Task result: the artifacts are copied to folder /artifacts of the bamboo agent.

3. “Command / Load release-image from artifacts”



Executable:

```
docker
```

Argument:

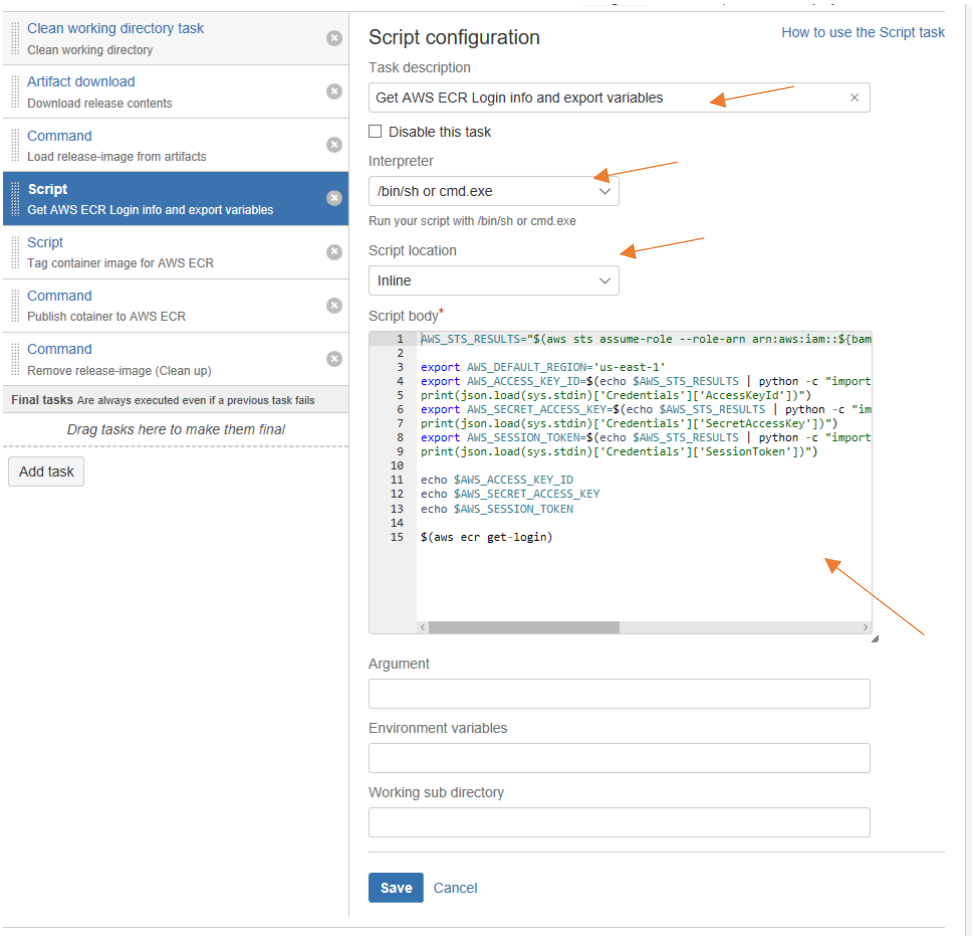
```
load --input /artifacts/vcp-netcore-webappdem01-release-image-artifact.tar
```

Task result

Since the artifact is a docker imager packaged in a .tar file, this task will load the image to the local docker registry of the bamboo agent.

4. “Script / Get AWS ECR Login info and export variables”

For pushing a container image to AWS ECR we are going to use the “AWS command line client” installed in the bamboo agent by default. To use the aws client we need the AWS logion tokens, this task will get that information.

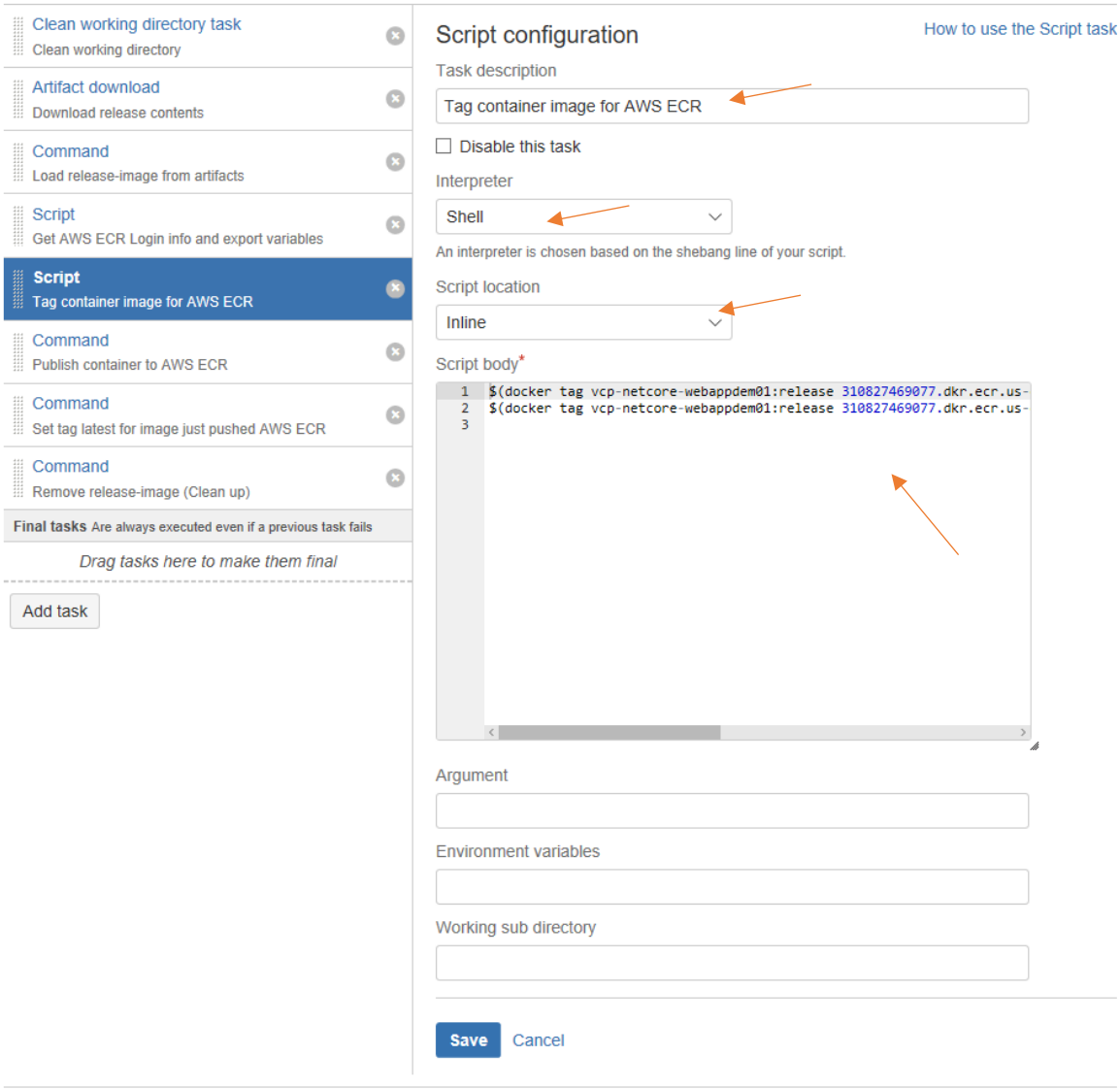


Script body

```
AWS_STS_RESULTS="$(aws sts assume-role --role-arn
arn:aws:iam::${bamboo.awsAccountNumber}:role/${bamboo.awsECSRole} --role-session-name
'compassSession')"

export AWS_DEFAULT_REGION='us-east-1'
export AWS_ACCESS_KEY_ID=$(echo $AWS_STS_RESULTS | python -c "import sys, json
print(json.load(sys.stdin)['Credentials']['AccessKeyId'])")
export AWS_SECRET_ACCESS_KEY=$(echo $AWS_STS_RESULTS | python -c "import sys, json
print(json.load(sys.stdin)['Credentials']['SecretAccessKey'])")
export AWS_SESSION_TOKEN=$(echo $AWS_STS_RESULTS | python -c "import sys, json
print(json.load(sys.stdin)['Credentials']['SessionToken'])")

echo $AWS_ACCESS_KEY_ID
echo $AWS_SECRET_ACCESS_KEY
echo $AWS_SESSION_TOKEN

$(aws ecr get-login)
```

IMPORTANT: The script uses some variables (in blue) defined for this “deployment environment”. We will show how to o setup those variables later in this document.

5. “Script / Tag container image for AWS ECR”

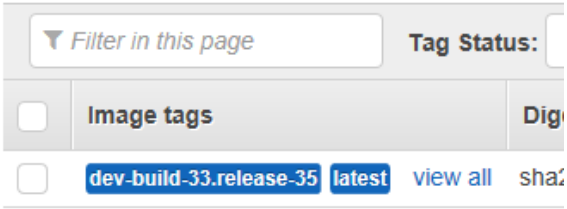


Script body:

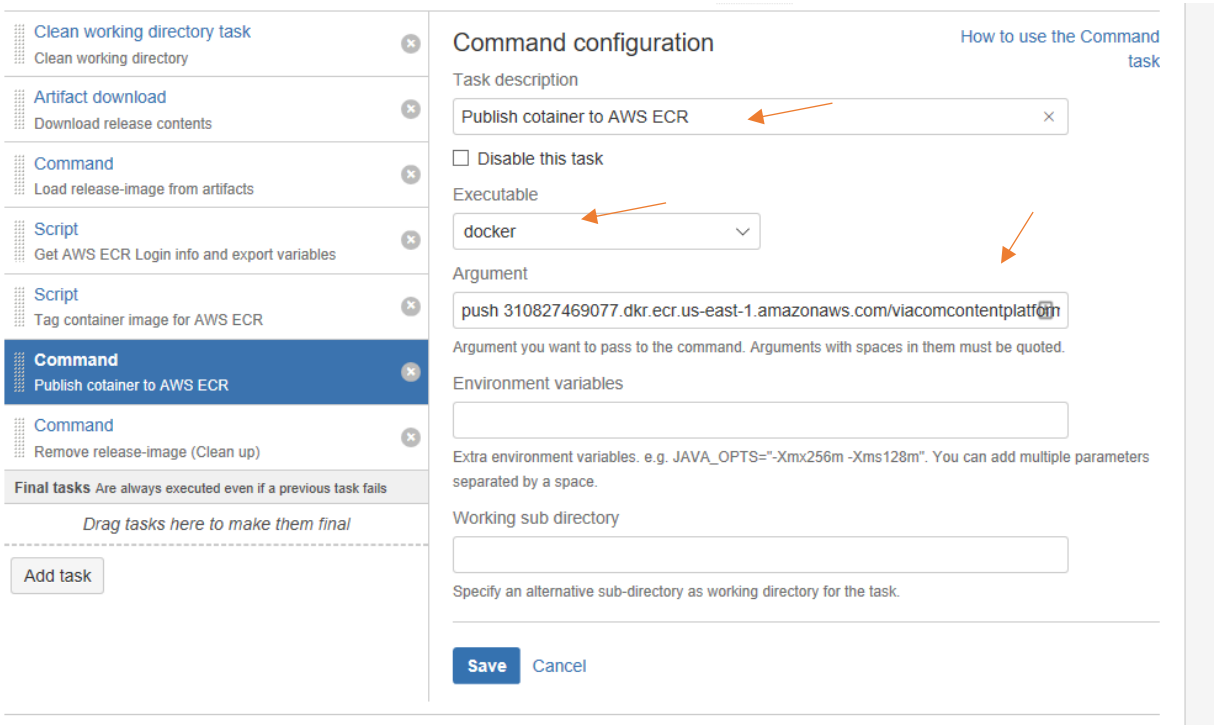
```
$(docker tag vcp-netcore-webappdem01:release 310827469077.dkr.ecr.us-east-
1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:${bamboo.deploy.version})

$(docker tag vcp-netcore-webappdem01:release 310827469077.dkr.ecr.us-east-
1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:latest)
```

Task result:

The Docker container image has been tagged using the version value and “latest” tag, so you can identify it on the image registry of AWS ECR.



6. “Command / Publish container to AWS ECR”



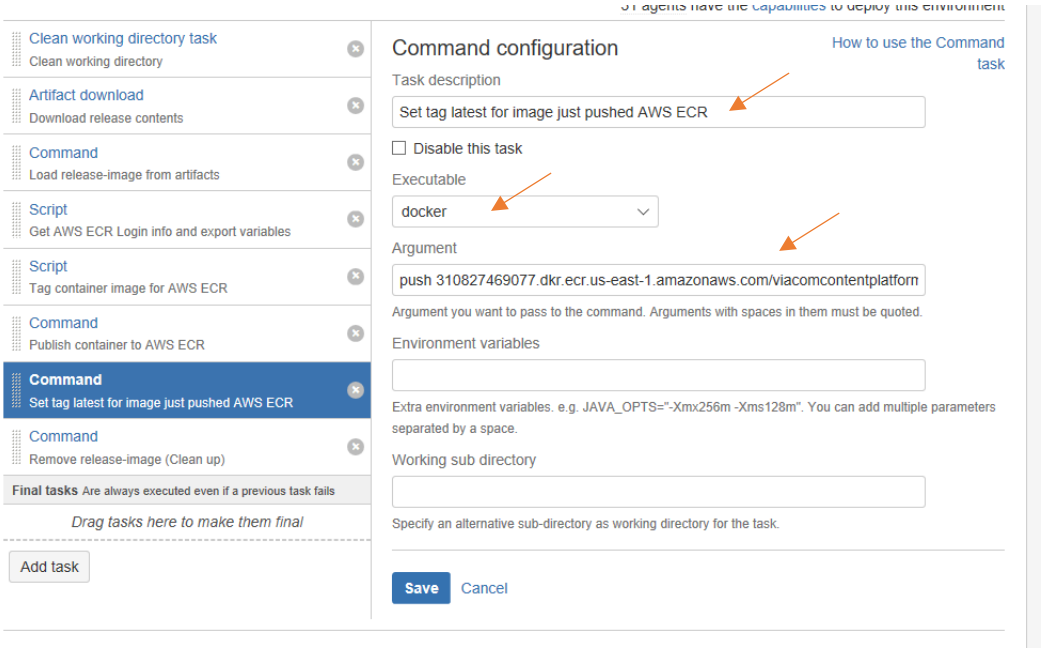
Argument:

```
push 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:${bamboo.deploy.version}
```

Task result

The container image is pushed from the bamboo agent to the container registry in AWS ECR.

7. “Command / Set tag latest for image just pushed AWS ECR”

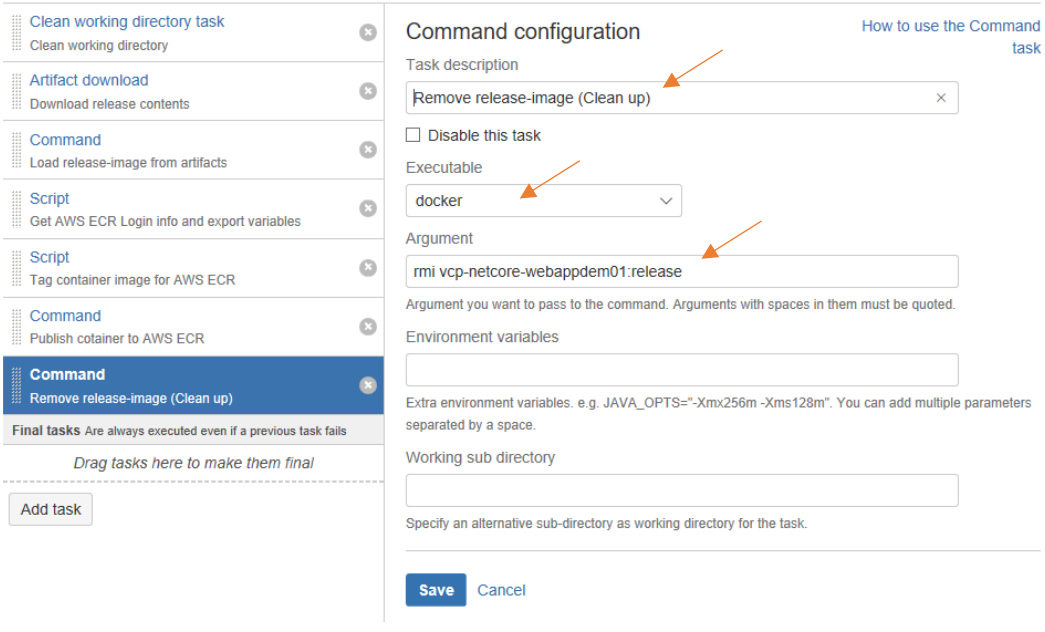


Argument

```
push 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:latest
```

Task result: This way the tag “latest” is set to the previous container image just pushed to AWS ECR

8. “Command / Remove release-image (Clean up)”



Argument: `rmi vcp-netcore-webappdem01:release`

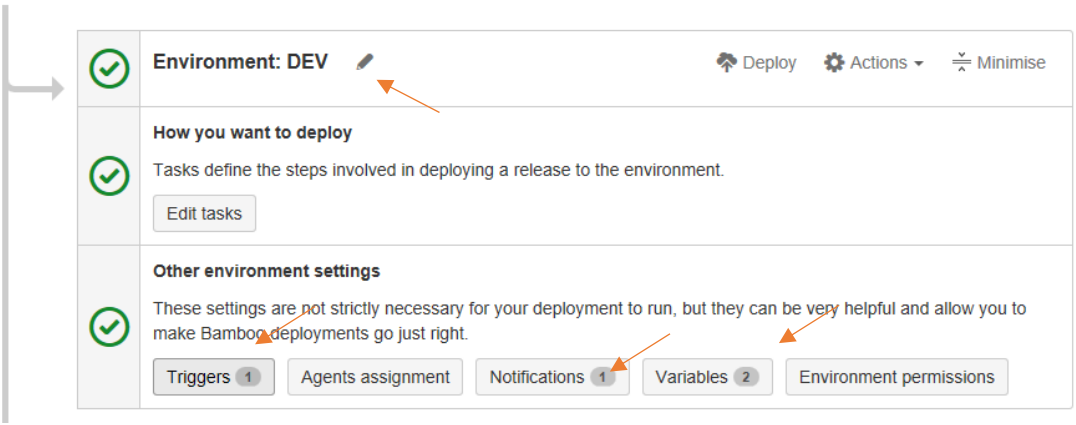
Finishing environments setup

So far, we've created a deployment project for the master branch, one environment (DEV) and the list of tasks to be executed for that environment during deployment. Now we need to setup additional parameters and variables around the environment.

Deployment triggers

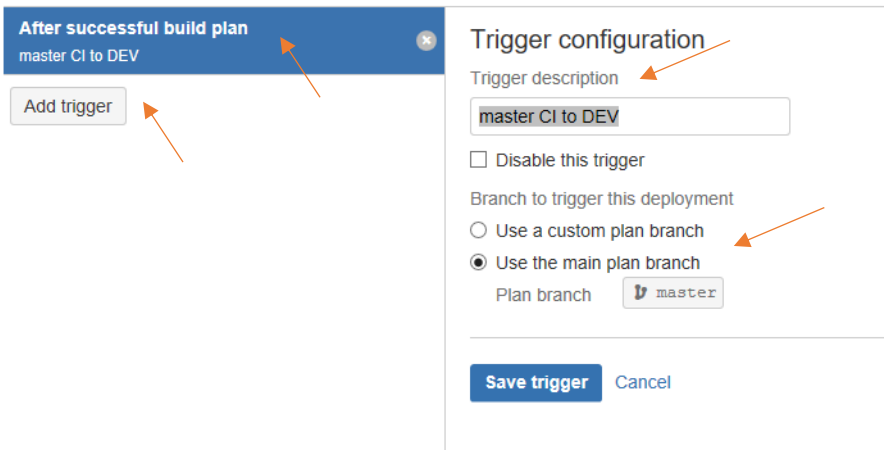
The triggers commands Bamboo to execute the deployment for a specific environment. In this example, we will setup one trigger for the master-branch/DEV environment:

1. Edit environment and click on "Triggers"



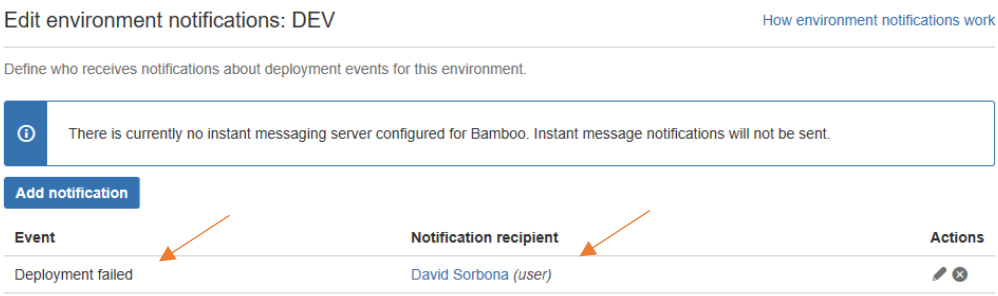
NOTE: When you are in edit mode, you can update also notifications and variables.

2. Add trigger type "After successful build plan / master CI to DEV"



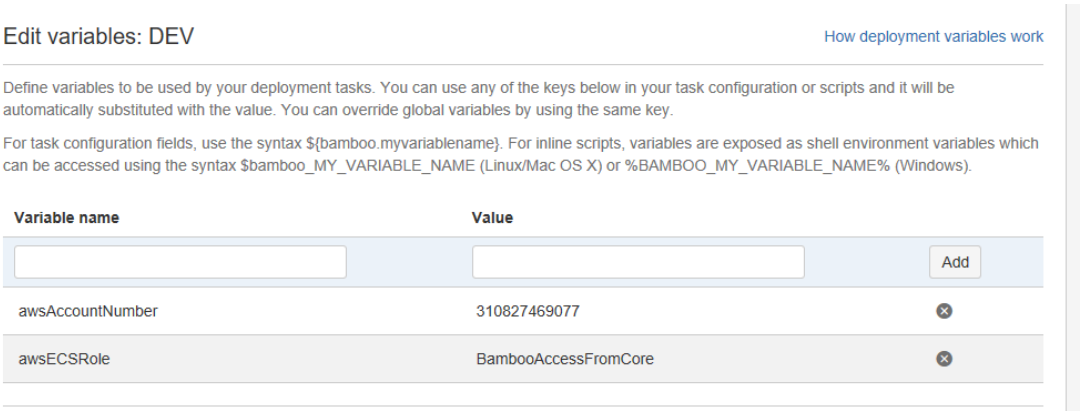
Deployment notifications

For this example, I will add just one notification when the deployment failed. The recipient should be the lead engineer or release manager:



Environment variables

The variables will be used by the tasks of the deployment process of this environment. For this example, we set two variables:

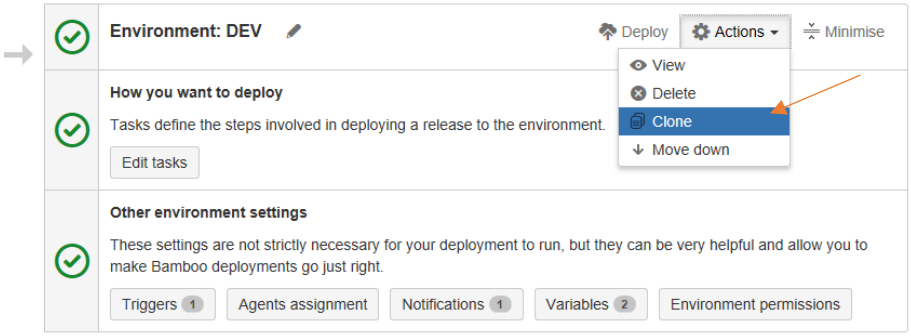


- awsAccountNumber 310827469077
- awsECSRole BambooAccessFromCore

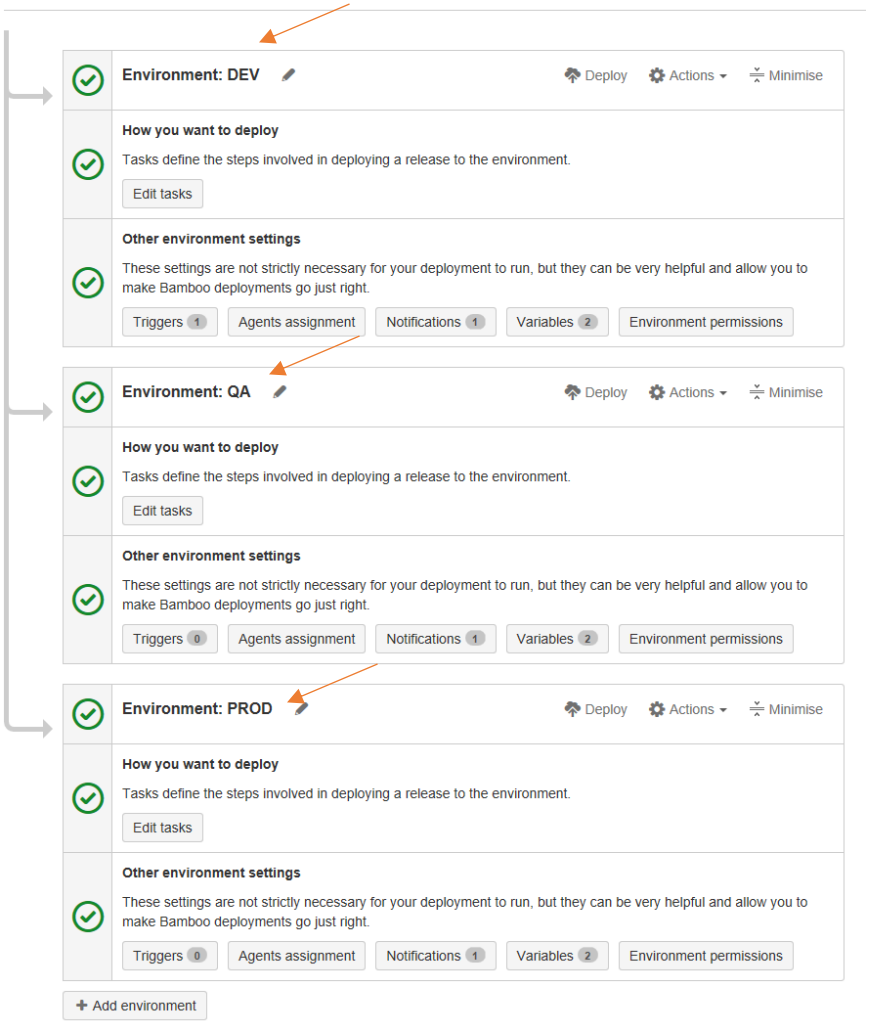
NOTE: This information is provided by the MCS team.

Create QA and PROD environments.

You can create the QA and PROD environment by cloning the DEV environment we've just created.

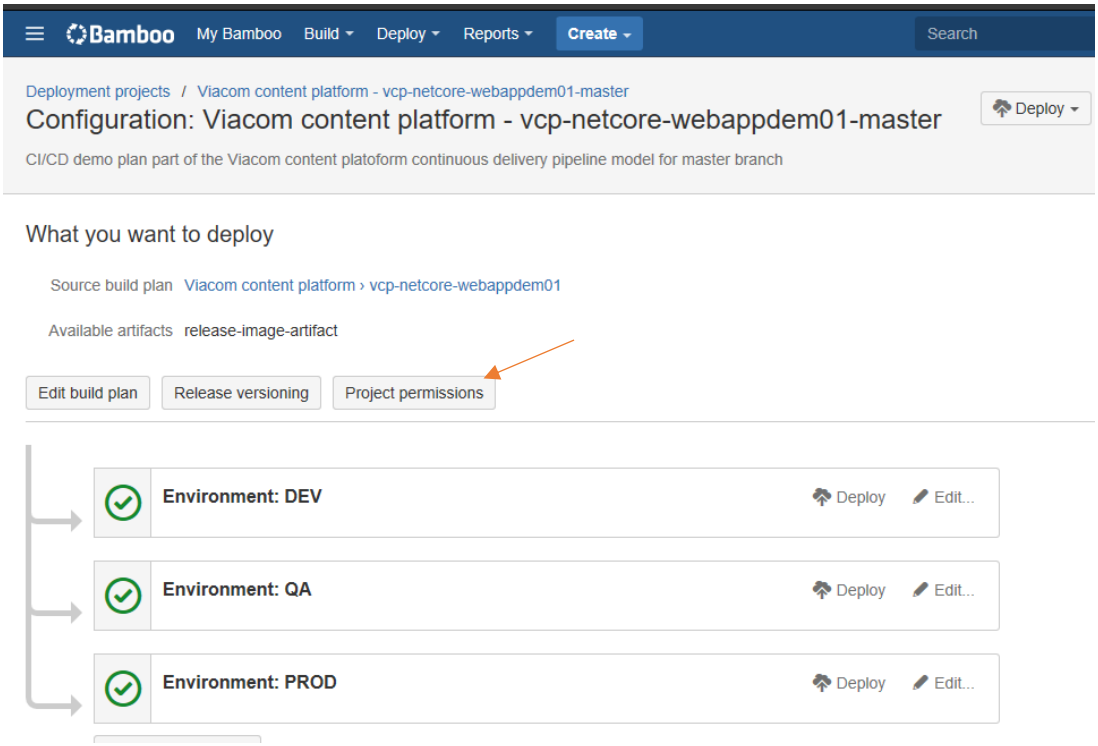


After that you have to update the variables and tasks as needed, for example the aws-image-tag could be different, the triggers, the notifications, the aws credentials, etc. At the end, you should see something like this:

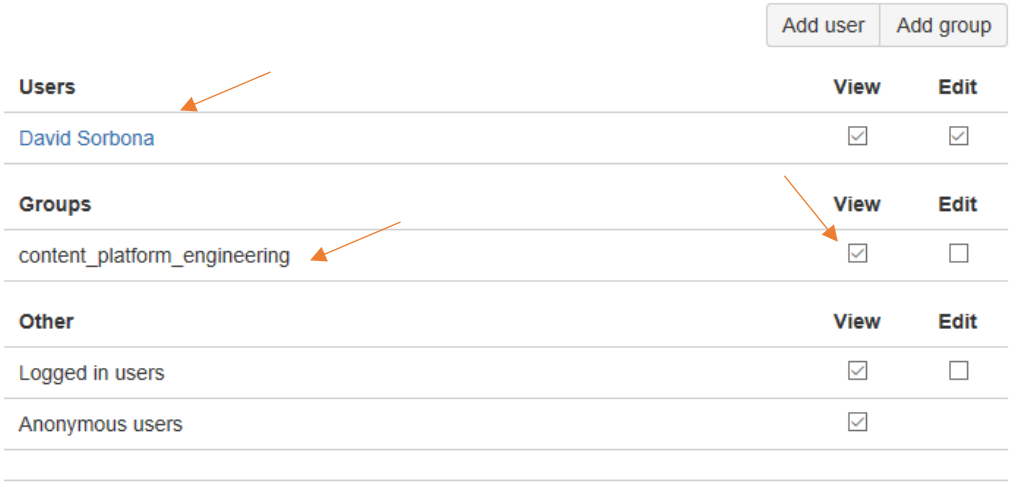


Set project permissions

The “content_platform_engirneering” crowd group should be able to see and run this deployment project and the release manager or lead engineer should be able to edit it.



Then set thepermission like this:

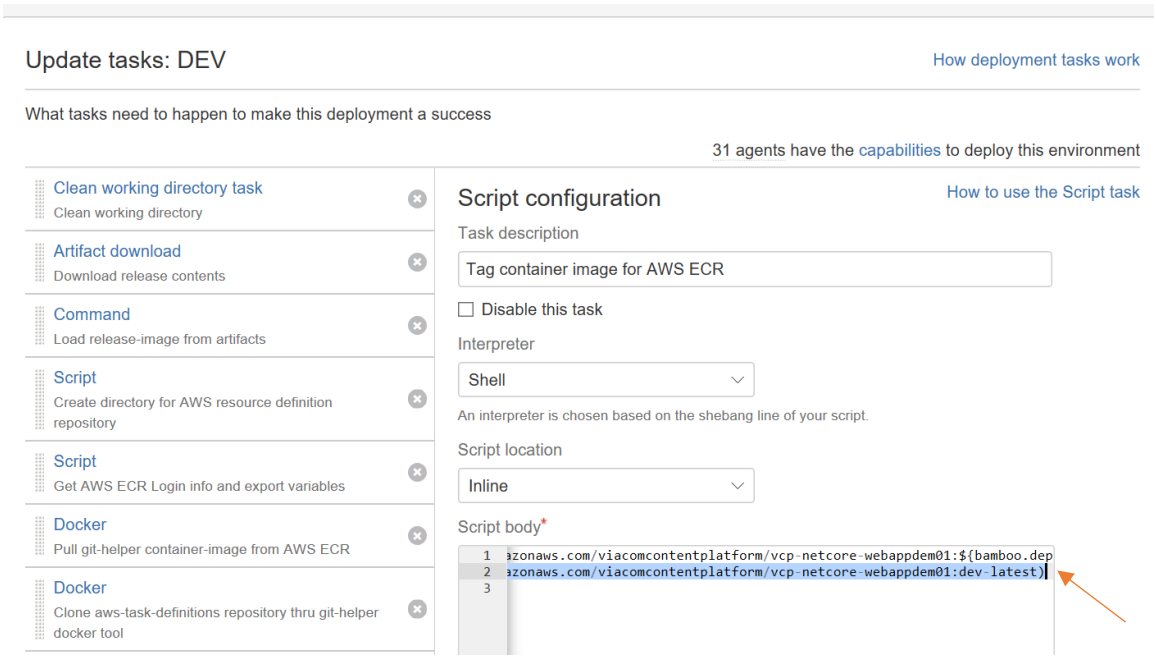


Deployment project setup for “development branch”

This deployment project is not mandatory but is good to have it for “nightly deployments”. Thank of this we will receive alerts of possible deployment issues in advance.

The steps to setup this deployment project are the same as explained before, just keep in mind the following differences:

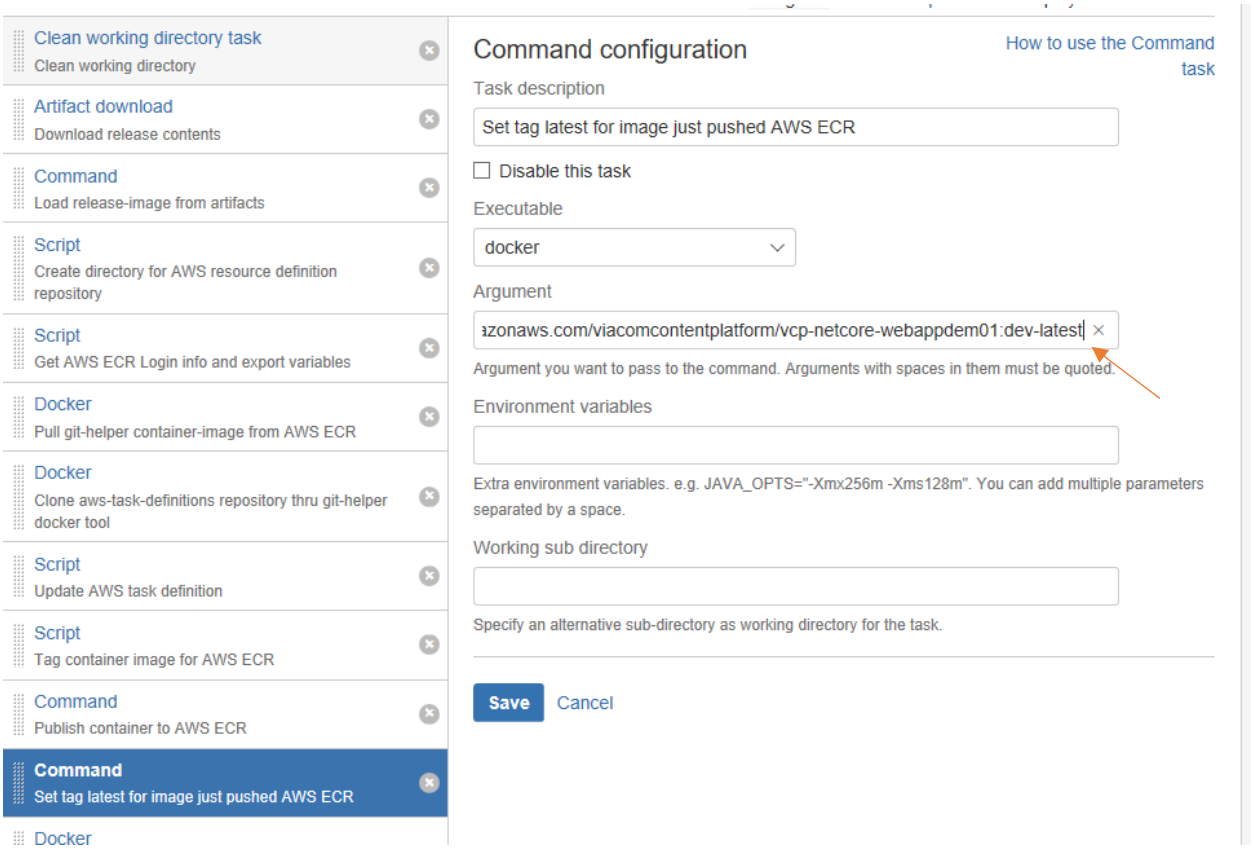
- This project has only one environment: “DEV”
- Select “development” as build plan.
- Release versioning: “**dev**-build-${bamboo.buildNumber}.release-1”
- In step “Script / Tag container image for AWS ECR” use another tag to identify the docker images. In this example we will use the tag “dev-latest”. This way we avoid tag collisions between DEV and PROD builds.



Script body:

```
$(docker tag vcp-netcore-webappdem01:release 310827469077.dkr.ecr.us-east-
1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:dev-latest)
```

- In step “Set tag latest for image just pushed AWS ECR” updates the tag to “dev-latest”



Argument:

```
push 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:dev-latest
```

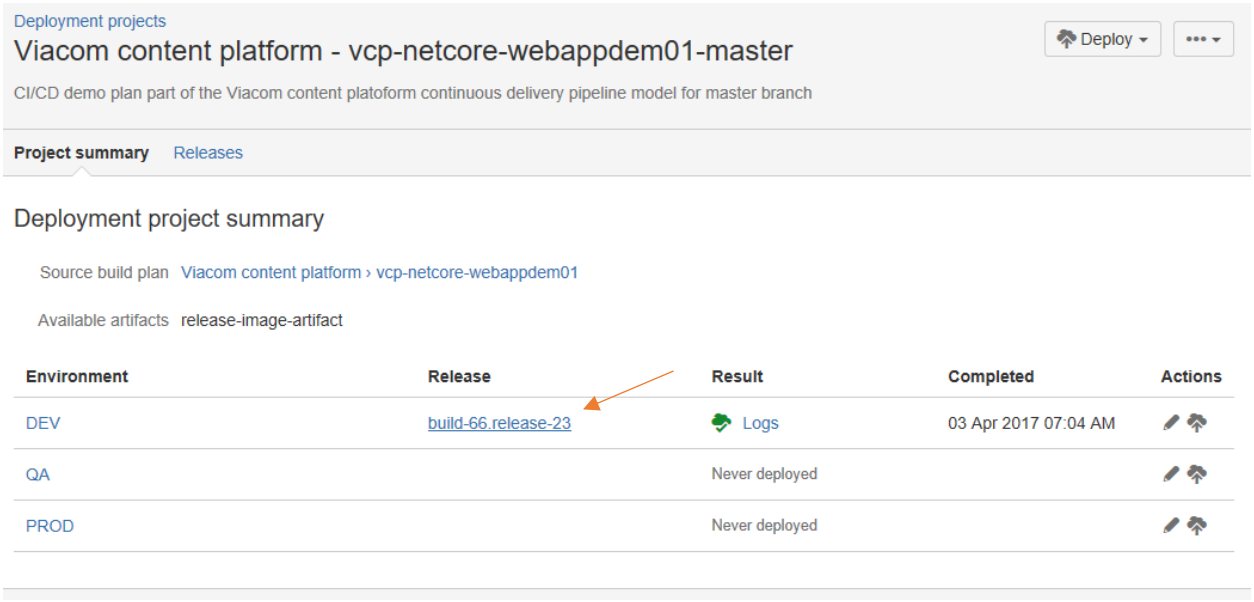
At the end, from the deployment project dashboard you should see two deployment projects: one for the nightly deployments (nightly flights?) and other for planned releases (planned flights?)

Working with deployments

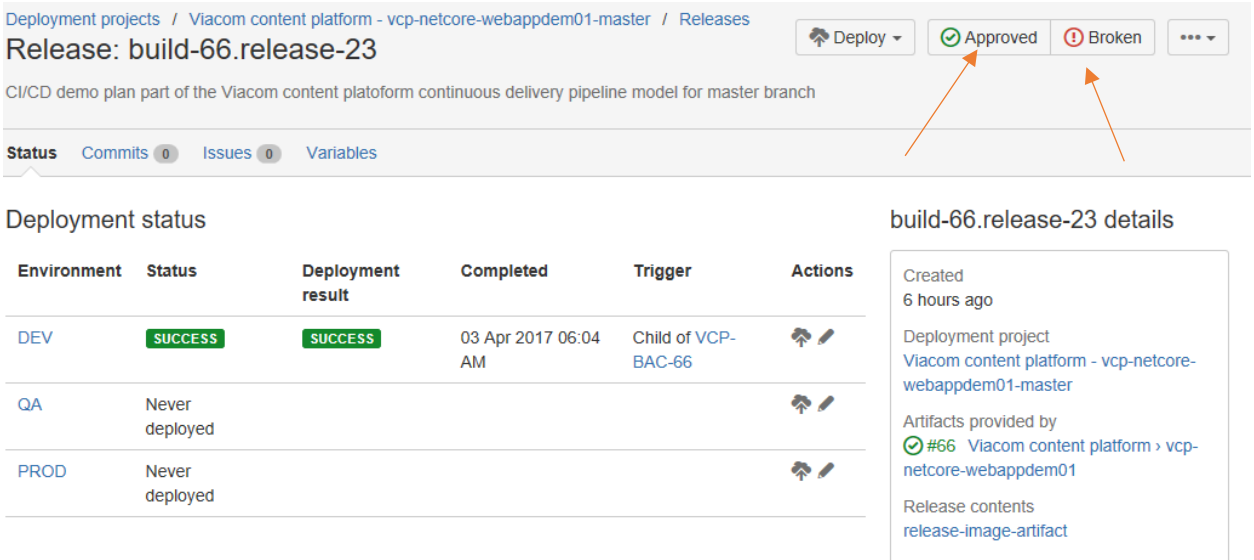
Once the deployment to an environment is completed the “release manager” or the “lead engineer” should flag the deployment as “approved” or “broken”. If the deployment crashes and burns, a rollback should be redeployed to the environment if needed.

Flag deployment as approved or broken

1. From the deployment project dashboard, go to the “last deploy status” page:



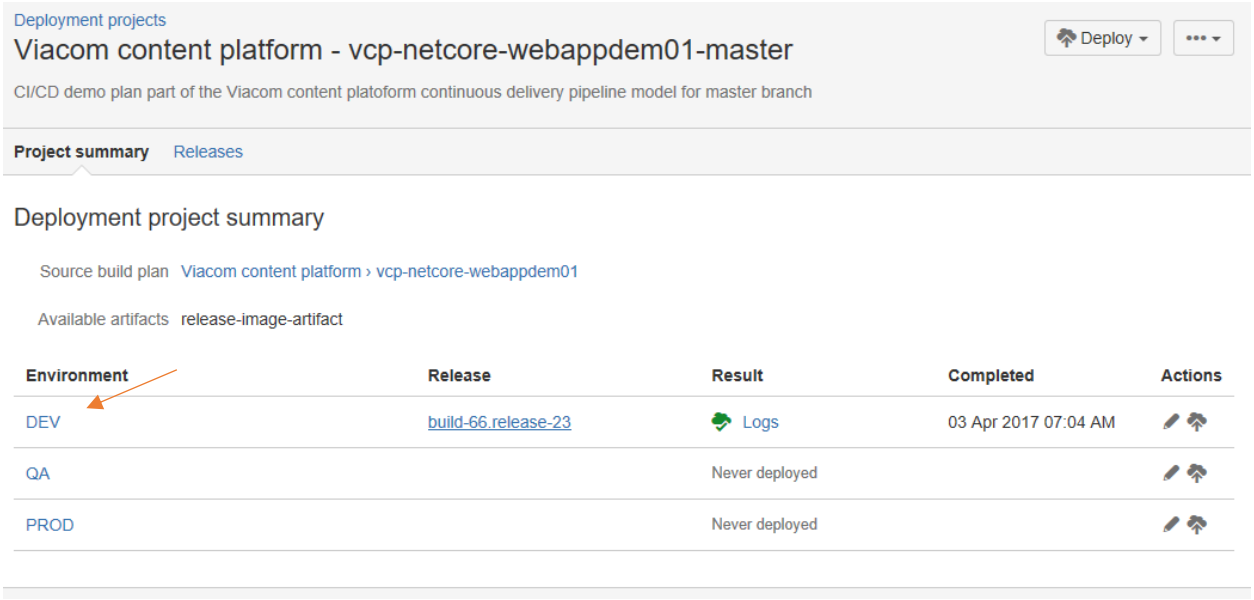
2. Flag the deployment as Approved or Broken as needed



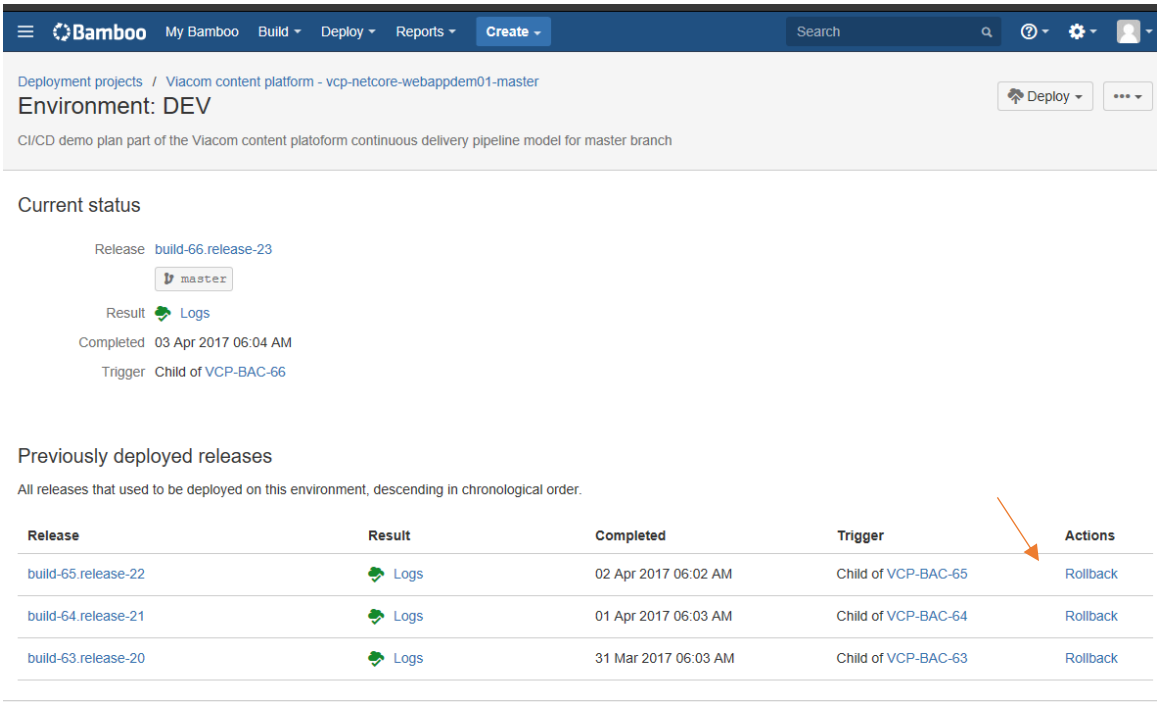
Rollback deployment

WARNING: For production environment contact first the “engineering board”.

1. From the deployment project dashboard, go to the “environment status” page:

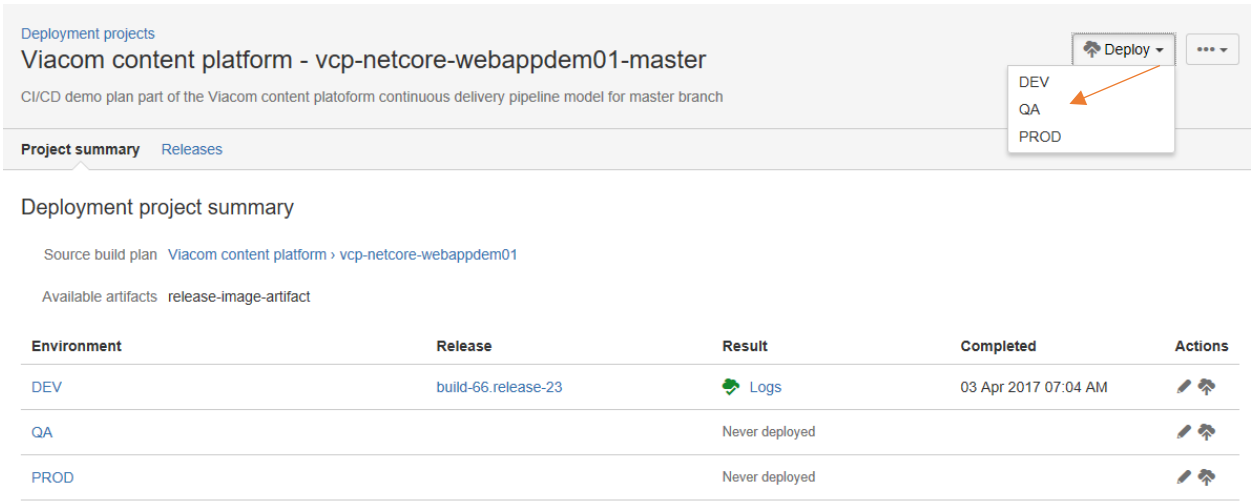


2. Rollback to the release you need:

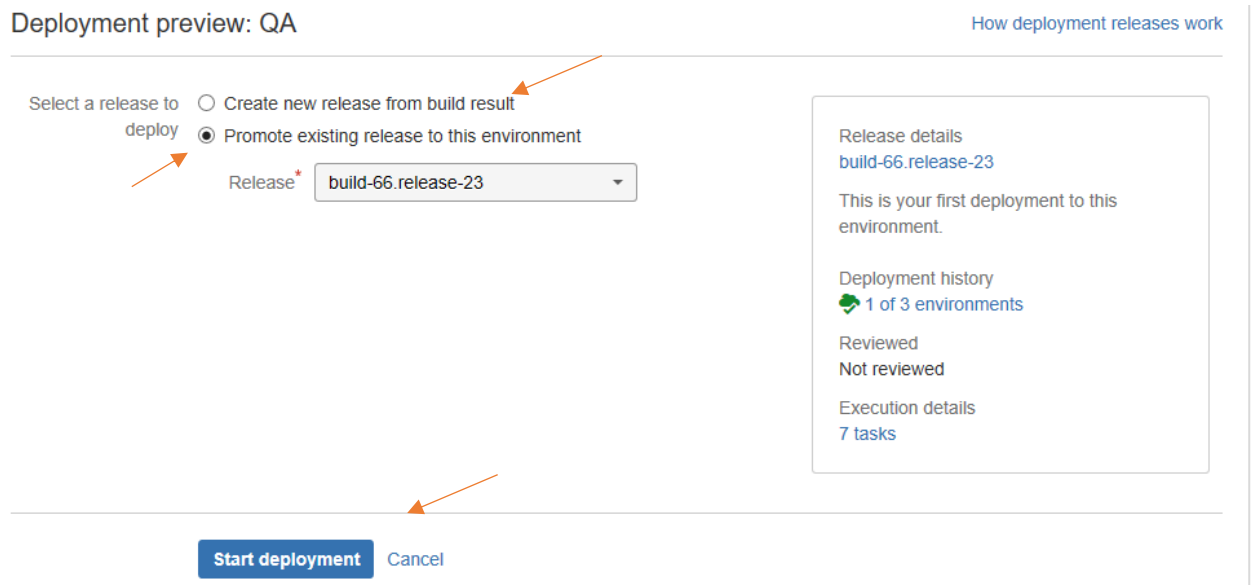


Manual deployment

To manually deploy a release, for example in QA or PROD you have to go to the deployment project dashboard, click on Deploy and select the environment:



Then you can promote an existing release or create a new release as needed:



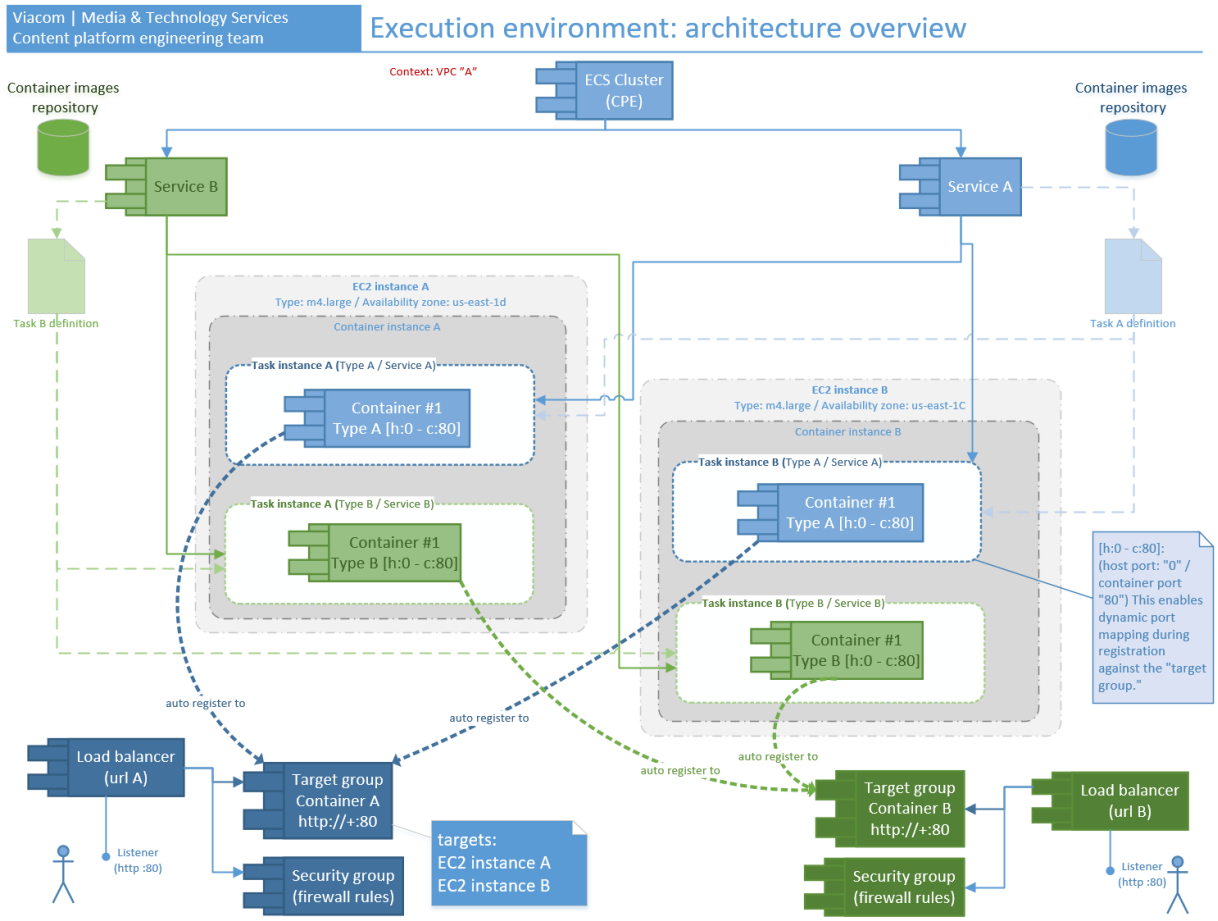
Setup execution environment in cloud provider

Introduction

Up to now we've just hosted our docker "release-image" in the Viacom private container registry in AWS ECR. This section will explain how to setup the execution environment in AWS and update the Bamboo deploy project to command AWS to deploy the new container-image version.

Architecture overview

The execution environment in AWS depends in several AWS resources that have to work together to provide the service needed. The following diagram shows the architecture of the environment used by the POC application described in this document, the architecture could be more complex depending the context and the needs of the application.

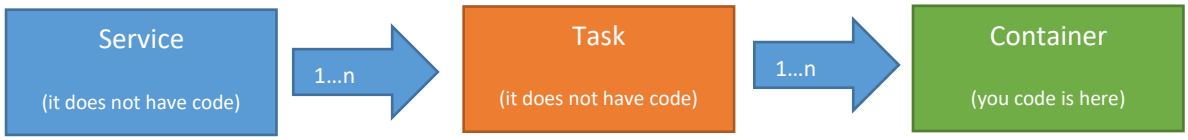


NOTE: There are more resources that were not included in the diagram.

NOTE 2: Yes it looks complicated at first glance but follow me, there is no magic here, all the concepts are well known.

Let me explain a little bit the context of this:

In AWS terminology a “Task” is an abstraction that groups one or more container instances running simultaneously, and a “Service” is another abstraction used to cluster one or more “Tasks” of the same type.



Under the hood the containers belonging to a “Task” will run in a “EC2 instance” (in other words, in a virtual machine). In addition, there will be an “AWS Load balancer” that will route traffic between the different “Tasks/containers”.

In our POC each “Task” has only one “Container” since our container “release-image” is autonomous, this means that it does not need anything else to run.

AWS resources and governance

To give you more context here a list of all the resources that are used by this POC and that you have to be aware of:

- VPC
- ECS cluster
- EC2 instances
  - Container instances
- Service definition
  - Task definition
- AWS Load balancer
  - Listeners
- Target group
- Security group
- Availability zone

ATTENTION!

- The resources in red can only be created and managed by the MCS team.
- Remember that the procedure explained later can’t be reproduced in the PROD environment, to create resource in PROD you have to provide a Cloud formation template.

Environments: prod vs. non-prod

The Viacom instance of AWS has two well defined type of environments: prod and non-prod. This procedure will show how to create the resource for the “NON-PROD” environment thru the AWS web portal, however for “PROD” the only way to create resource is by templates files called “cloud formation templates”. We will explain how to use cloud formation in a future version of this document.

The access to those environments is under the governance of the MCS team.

Pre-requirements

Before continue you will need:

- Access to the AWS web portal, account “vmn-platform-nonprod”
- One “ECS cluster” with at least 2 EC2 instances in two different availability zones (all in the same VPC)
- VPC ID. This defines the context where the resources are created.

For any of those you have to put in contact with the MCS team.

Create task definition

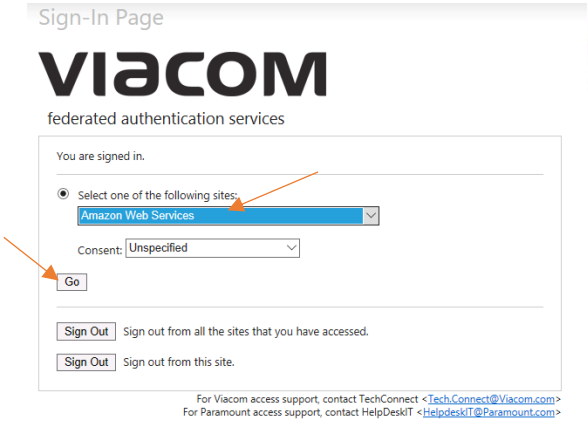
The first step is to create a “Task definition”, in essence it is a set of properties that describes the characteristics of the task, for example: it defines the amount memory/CPU that will be assigned to their containers, it defines the docker-images that will be used to create the container instances, etc.

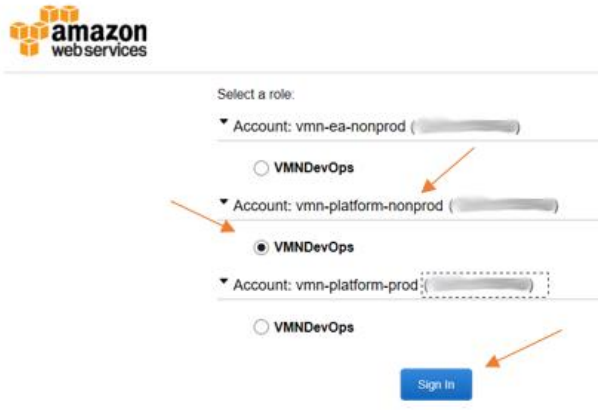
You can create the task definition via AWS Web portal or thru Cloud Formation. In any case, the objective here is to create a json file with all the properties of the “Task”. We will use this json file as a template that will be consumed by our deployment projects in Bamboo, I will explain why of how later.

To create the task definition, follow the next steps:

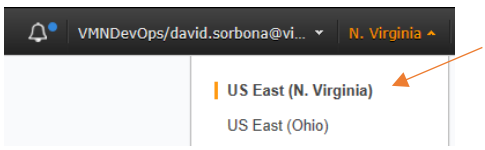
1. Log to AWS web portal with your AD credentials and select the account: “vmn-platform-nonprod”

   https://fs.viacomcloud.com/adfs/ls/IdpInitiatedSignon.aspx

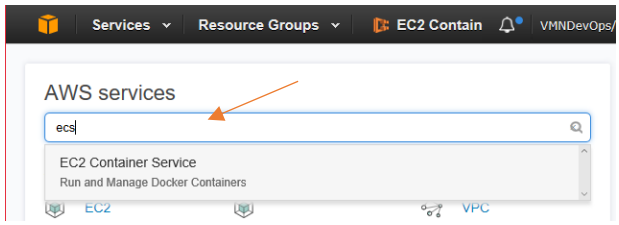


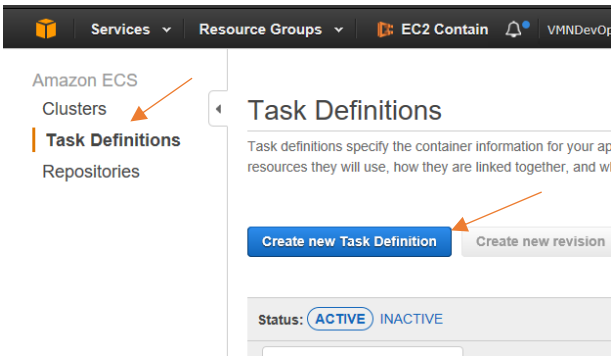


2. Make sure you are in “N. Virginia (us-east-1)”

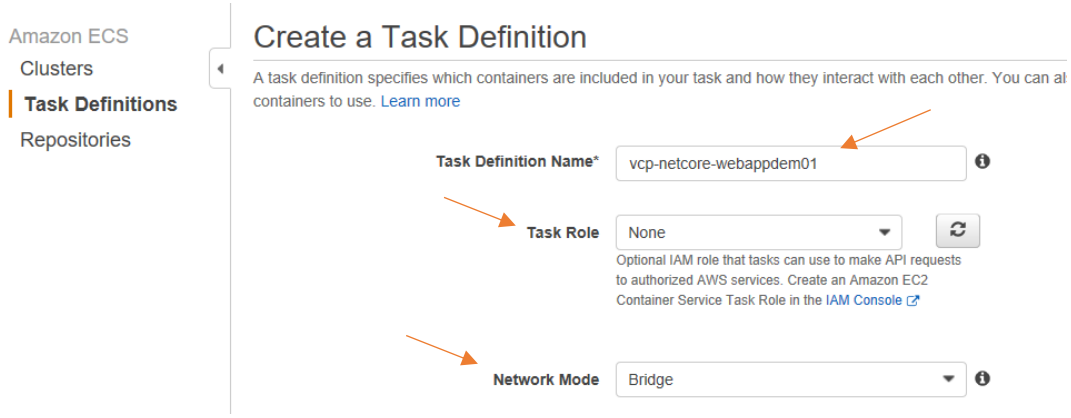


3. Go to “EC2 container service” sub portal, then o to “Task definitions” and then click on “Create new Task Definition”



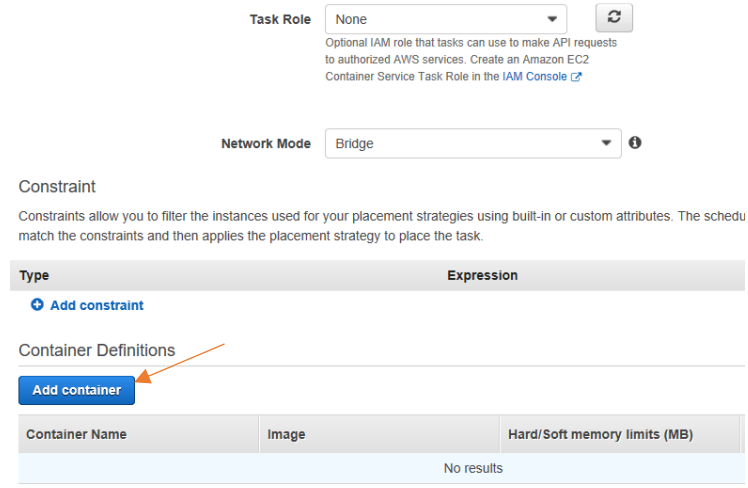


4. Completes the form as shown:

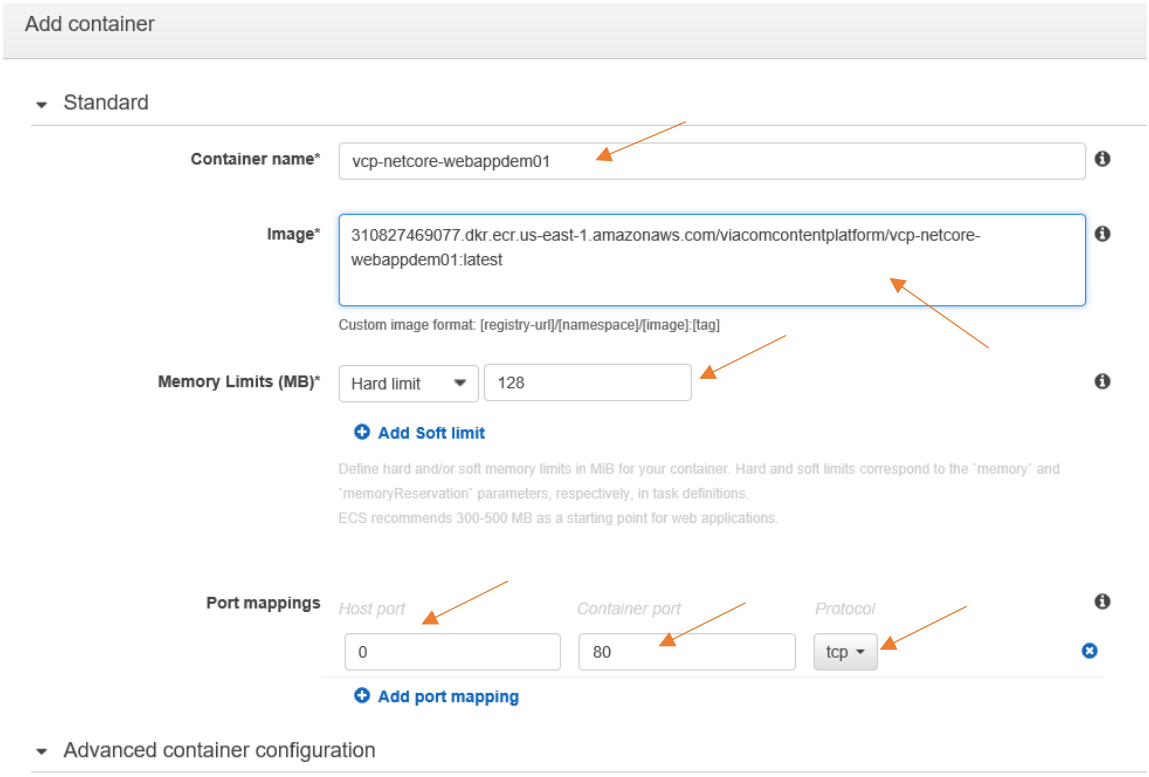


Naming convention: For the task definition name, use the same name of the application repository, in this example we will use “vcp-netcore-webappdem01”

5. Click on the “Add container” button:



6. Completes the “Standard” section as follows:



Container name: vcp-netcore-webappdem01 (use the same name of your code repository)

Image: 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:latest

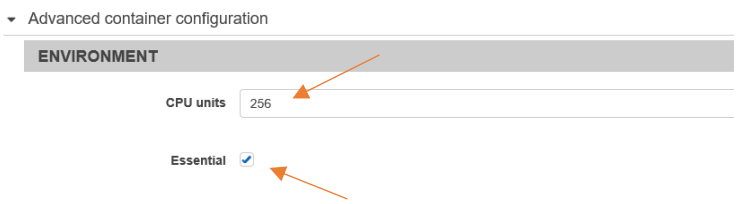
NOTE: the image is the AWS ECR URI where your container “release-image” is hosted.

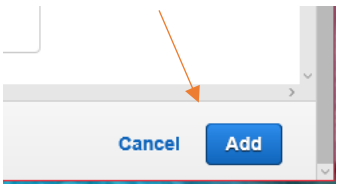
Memory Limit: Hard limit / 128

Port mappings: host: 0 / container: 80 / tcp

CRITICAL: the 0/80 configuration is use by AWS infrastructure to enable dynamic port mapping. This means that the EC2 instance can dynamically use any port to establish a map to the port 80 of the container.

7. Completes the “Advanced container configuration” section as follows:

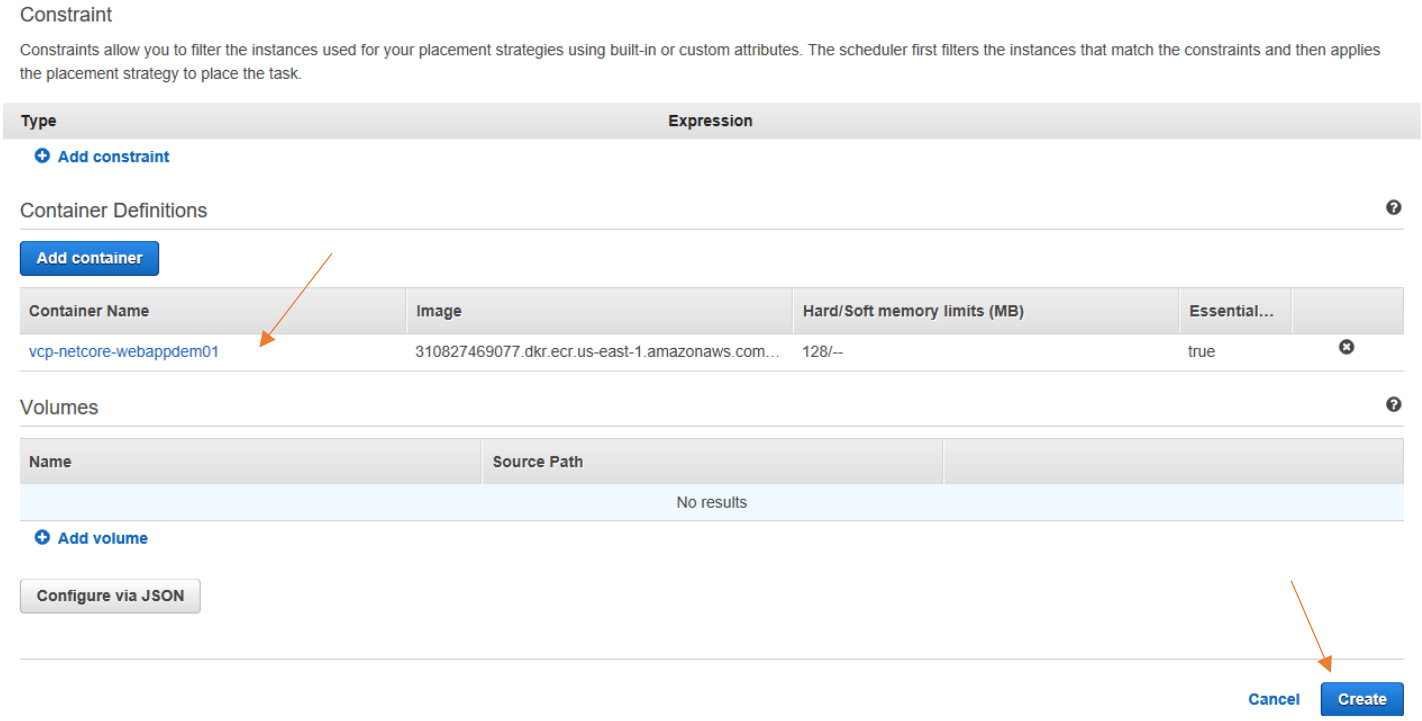




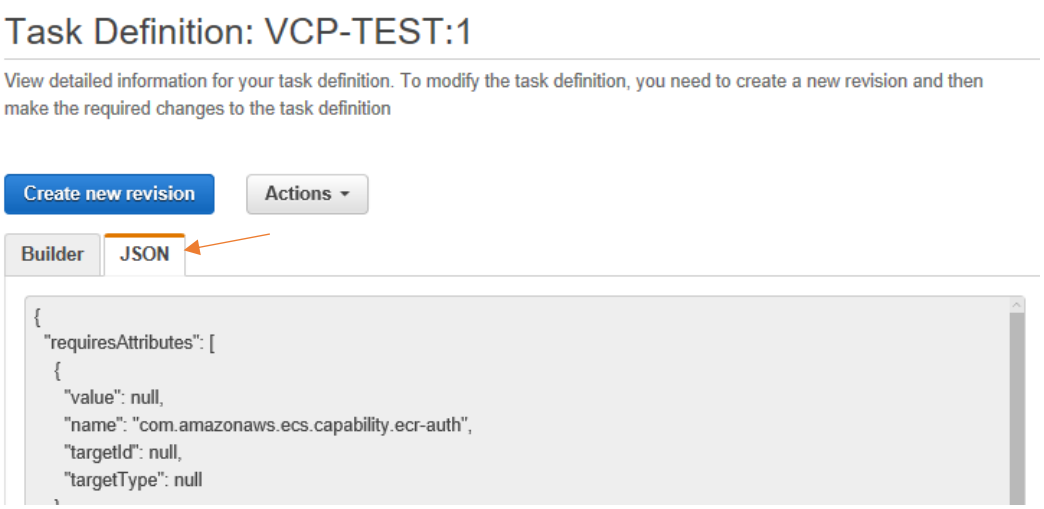
Leave the rest of the fields in blank and click on “Add”.

NOTE: more robust/complex configuration are possible but are out of scope of this document.

8. You should see a new container definition entry, finally click on “Create”



9. Go to the JSON tab and save the json document to a file.



Save this information in a file with the following naming convention

taskdefinition-[repository name]-template.json

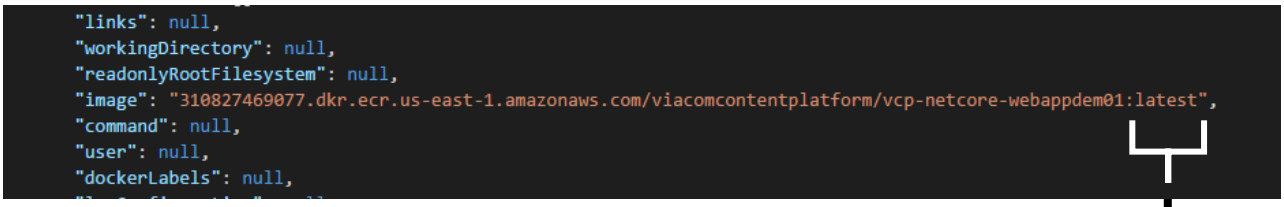
In this example the file is called: `taskdefinition-vcp-netcore-webappdem01-template.json`

Converting the task definition into a template.

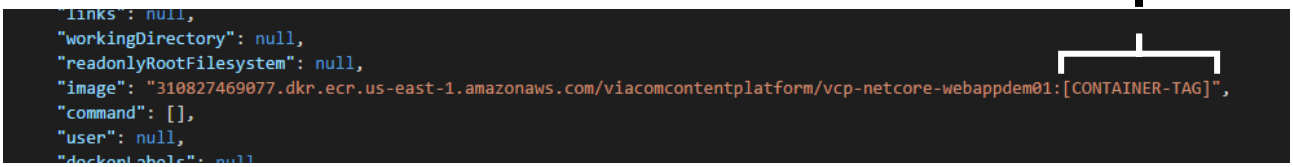
The Bamboo deploy process will use the task definition json file as a template to dynamically create new tasks definitions, later the new “AWS resource definition” will be sent to AWS API to request the deployment of new versions of our container “release-image”.

The last step is to substitute all variables of our json file with “string-tokens” that will be replaced by Bamboo on deploy time. In this example we are going to update only the container image name.

1. Open the json file and localize the property “image”



2. Replace the image tag “*latest*” by the token “*[CONTAINETR-TAG]*”



NOTE

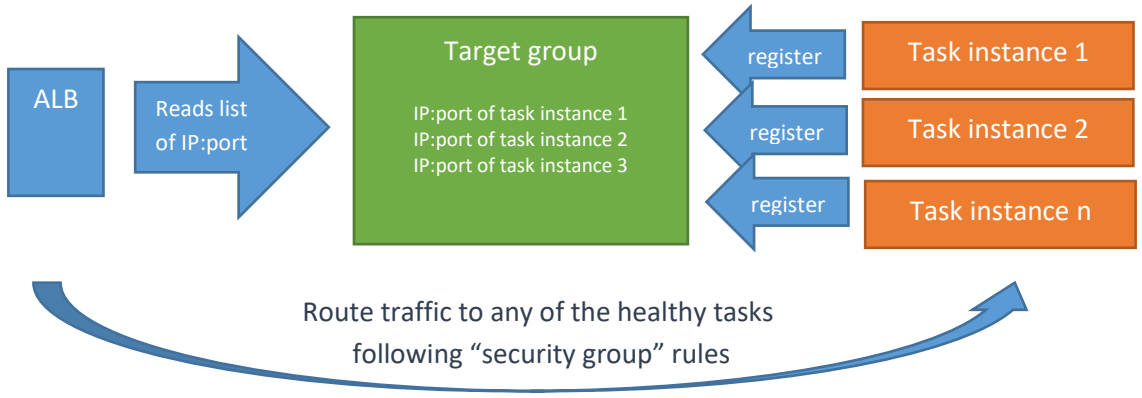
The same technique could be used to update other variables on deploy time, like the amount of memory, CPU or any other task/container property.

Congratulations! You have your “Task definition template” ready. Let’s create the other AWS resources.

Setup load balancer, target and security groups

The AWS load balancer will route the traffic to any of the tasks of our service. ALB (AWS Application Load Balancer) works together with target groups and security groups.

How it works? every “Task” that reaches a “healthy” status will register itself against a “Target group”, the ALB will route the traffic to any of the Task registered in the “Target group”. When a “Task” dies it is unregistered from the “Target group”, so the ALB stop sending traffic to it. In parallel, the security group acts a set of firewall rules.



There are different ways to create the load balancer, our approach will be: 1 load balancer per service.

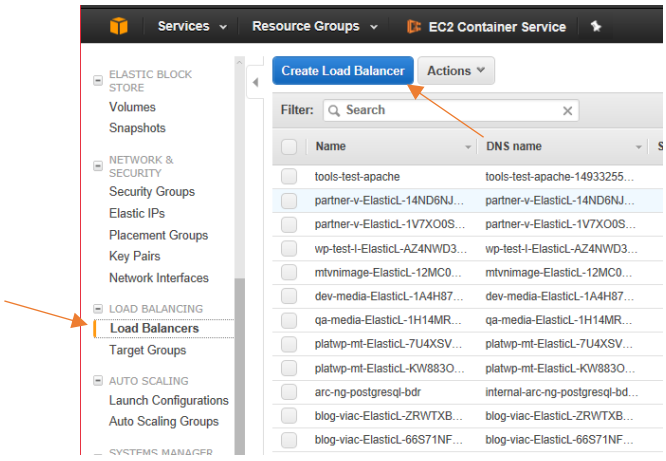
CRITICAL: Make sure you create all resources in the same VPC ID

To create the AWS Load balancer, follow this steps:

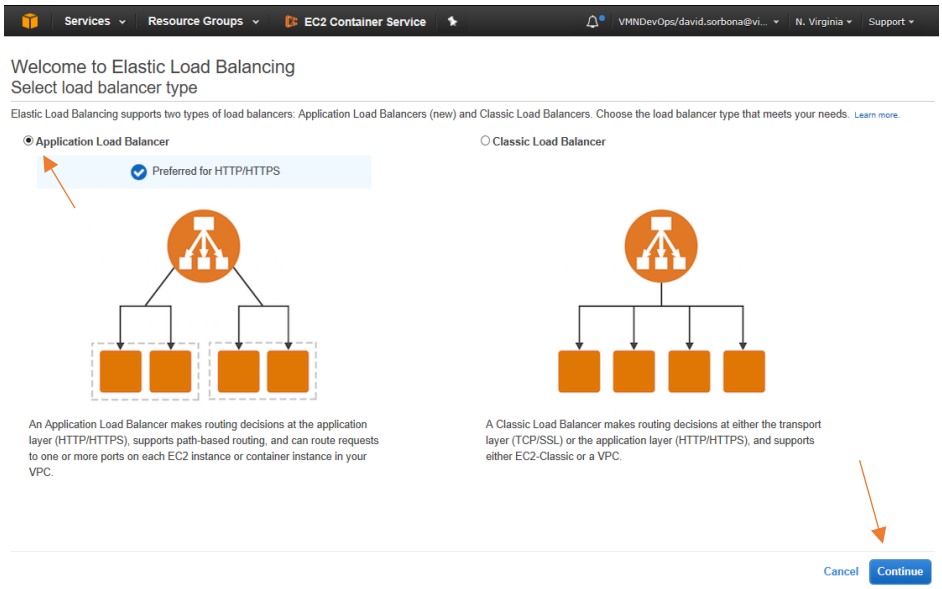
1. In the AWS web portal go to the EC2 dashboard for us-east-1 region:

   https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1

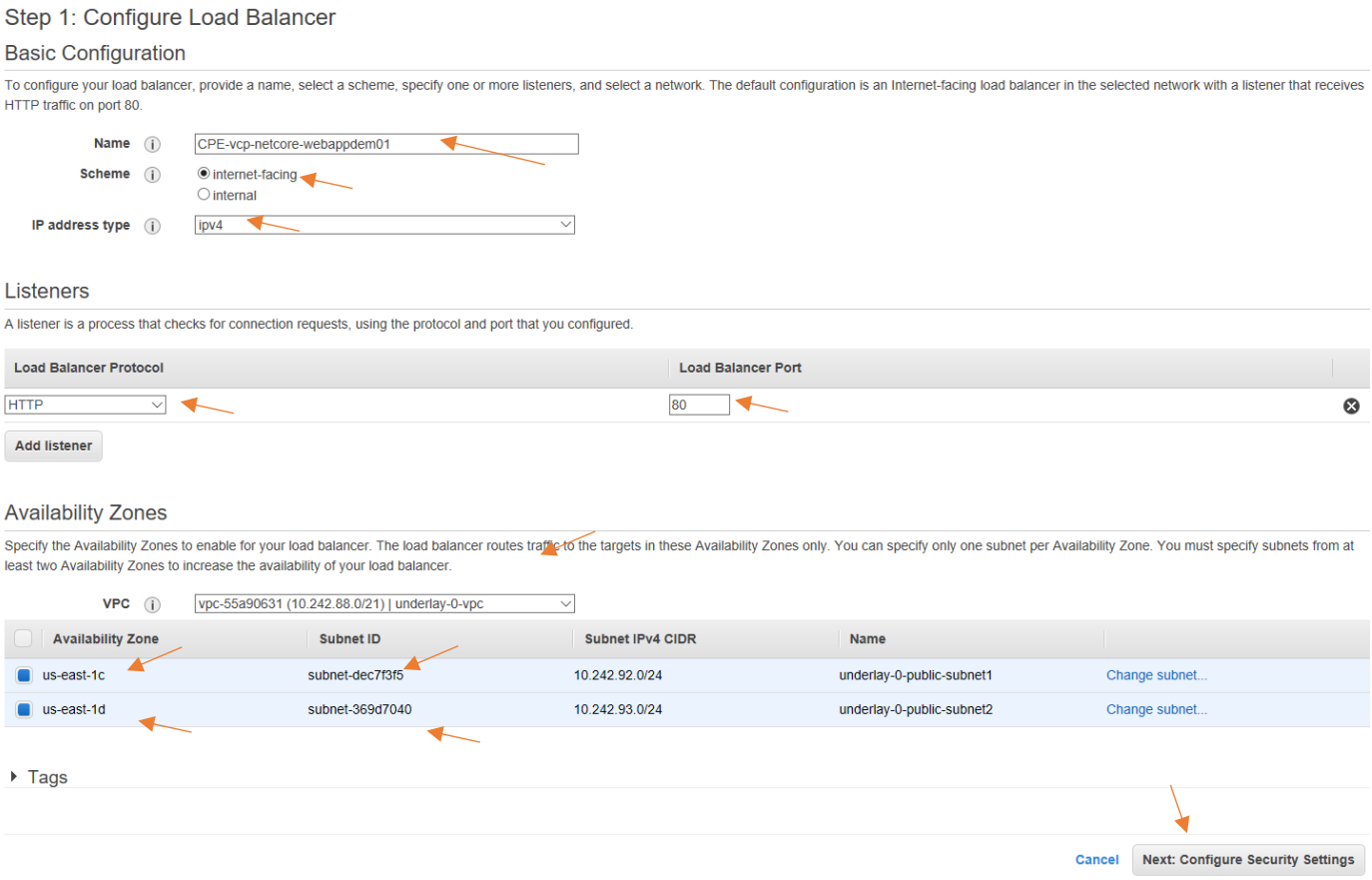
2. Go to the “Load Balancers” section and click on “Create Load Balancer”



3. Select the “Application Load Balancer” and click on “Continue”

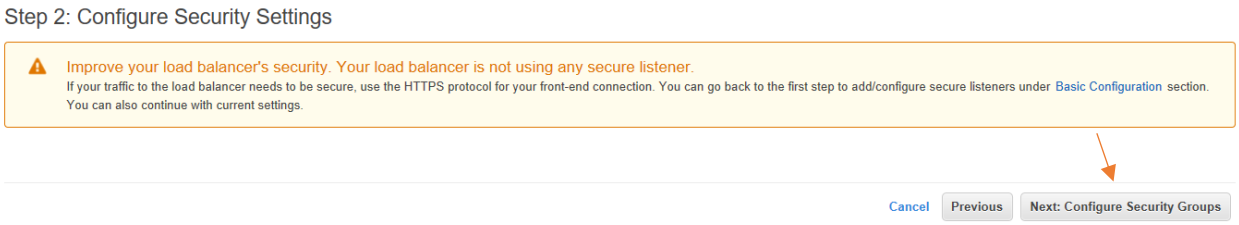


4. Fill the form as follows and click on “Next: Configure Security group”

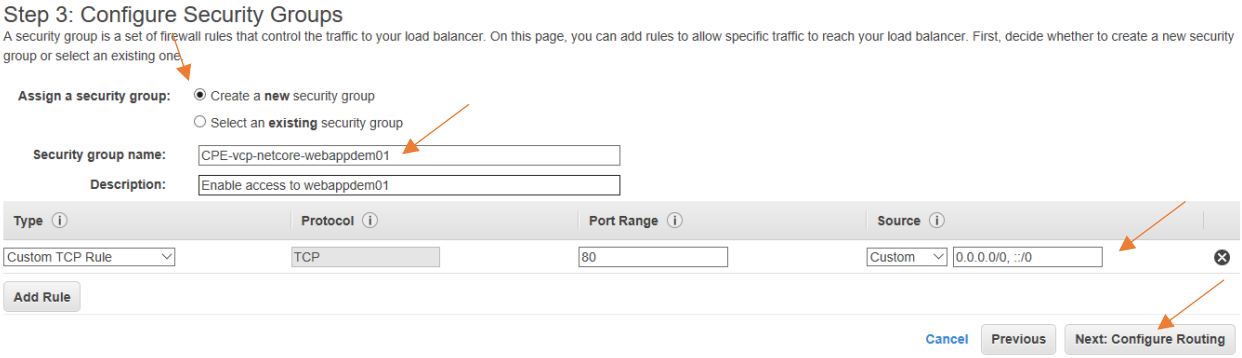


- **Name**: `[cluster name]-[repository name]`
  In our example “CPE-vcp-netcore-webappdem01”
- **Scheme**: internet-facing
- **IP address type**: ipv4
- **Local balancer protocol**: HTTP / 80
- VCP: vpc-55a90631 (CRITICAL!)
- **Availability zone**: us-east-1c (subnet /24) / us-east-1d (subnet /24)
  WARNING: Those two availability zones are the only ones that can be selected.

5. In our example we are not setting HTTPS, so we will skip the “Step 2”



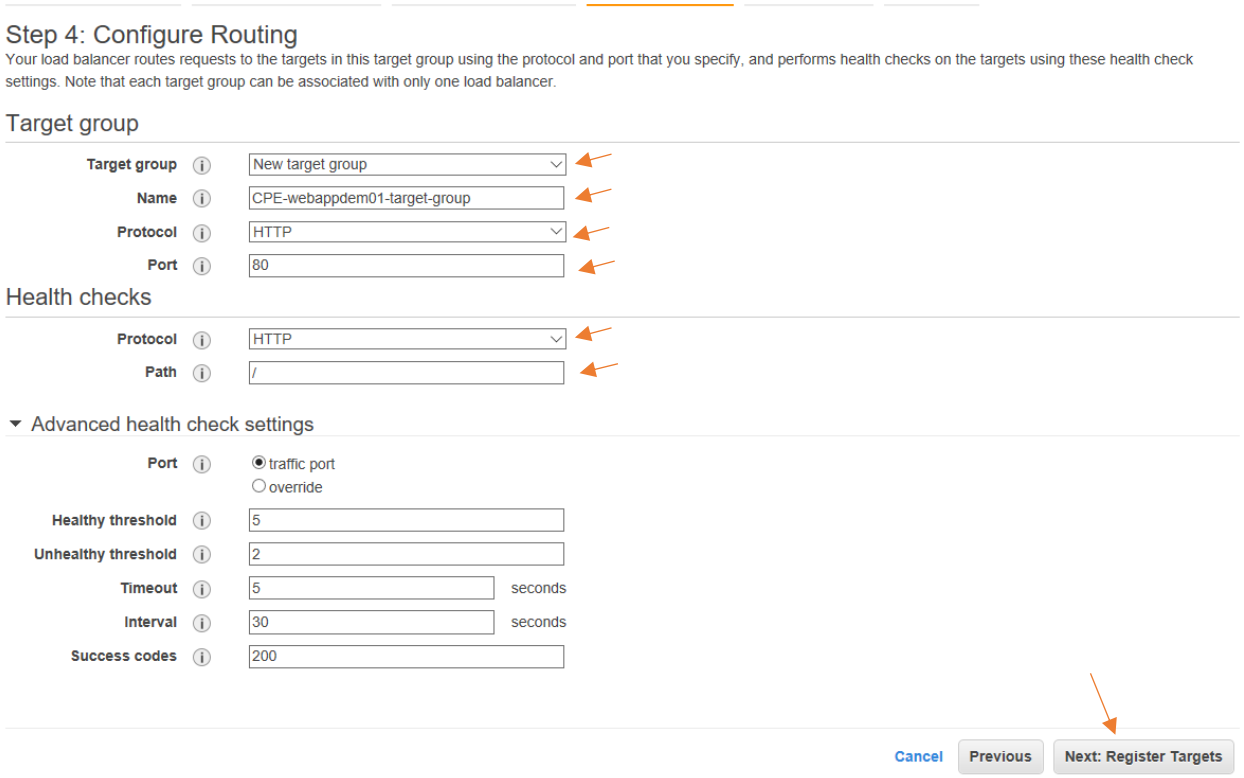
6. Let’s create a new “Security group” as follows and then click on “Next: Configure Routing”



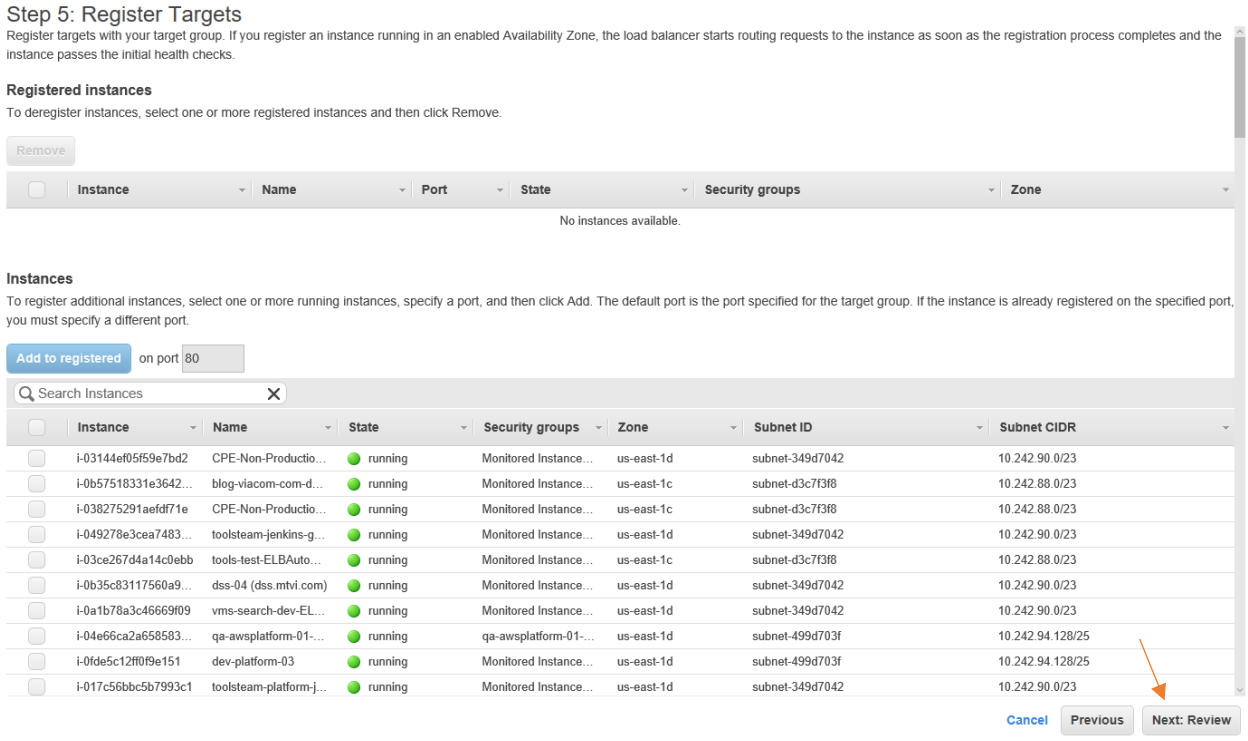
NOTE

In other scenarios it would be a good idea to share the same “Security group”. For doing that you have to create the “Security Group” before creating the Load balancer.

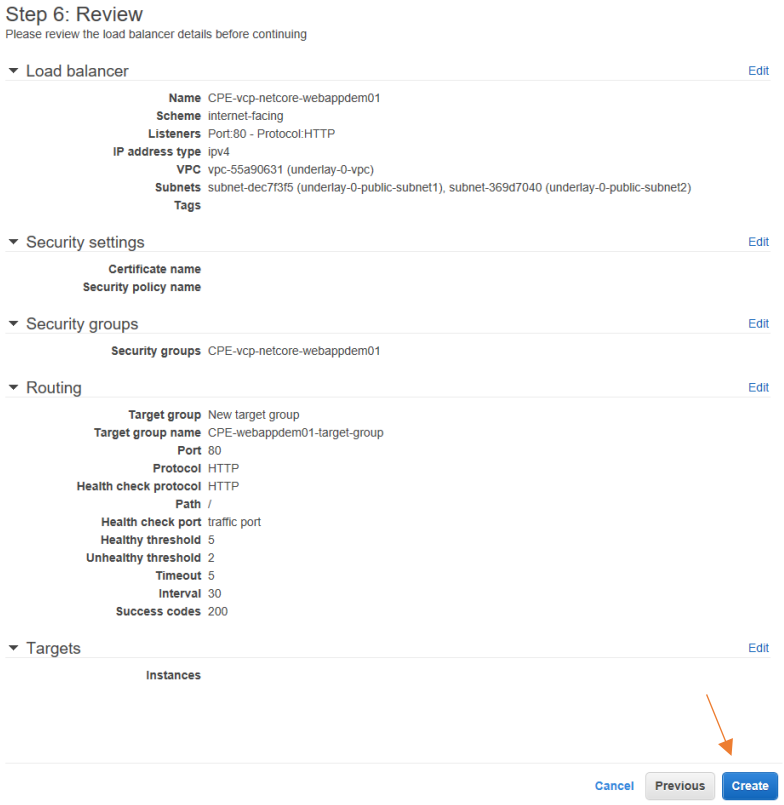
7. Let's creates a new "Target group" as follows and click on "Next: Register Targets"



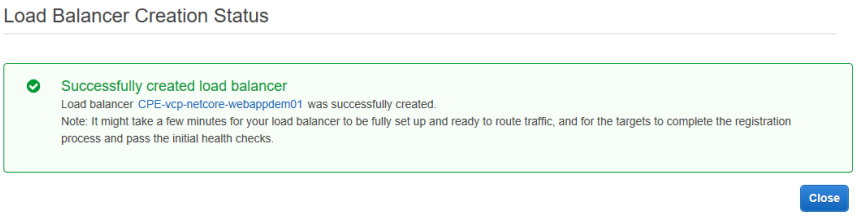
8. We are not going to explicitly register instances, so let's click on "Next: Review"



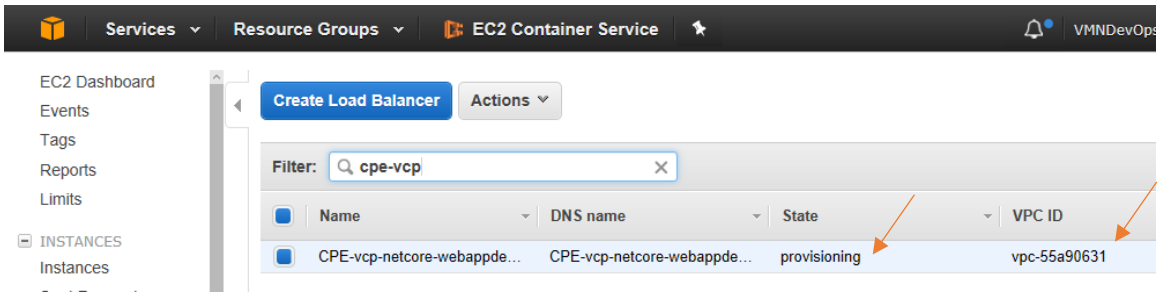
9. Review the configuration and click on “Create”



10. Done! You have created three AWS resources: a new “application load balancer” (ALB), a new “security group” and a new “target group”



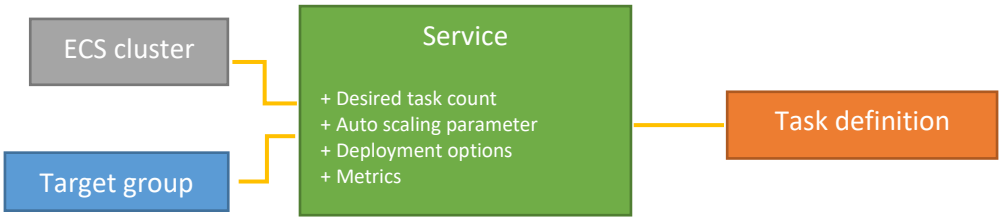
Note that the provisioning of this could take several minutes. When its state is “Active” it is ready to rock!



Congratulations! You have a ALB ready to route traffic to your service, so let’s create the service!

Setup service

In AWS a “Service” is just an abstraction that links three resources: the “ECS cluster”, the “Task definition” and the “Target group”. Additionally, the “Service” defines how many task should be running, deployment options and auto scaling parameters.

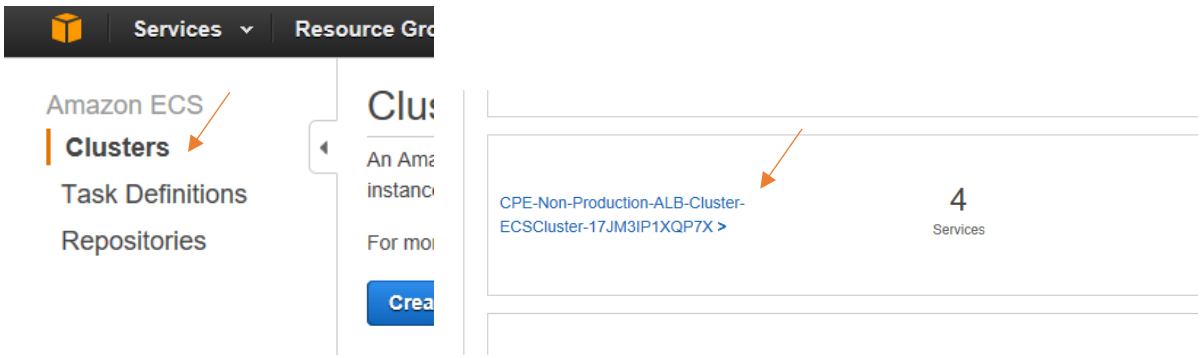


To create the service follows the next steps:

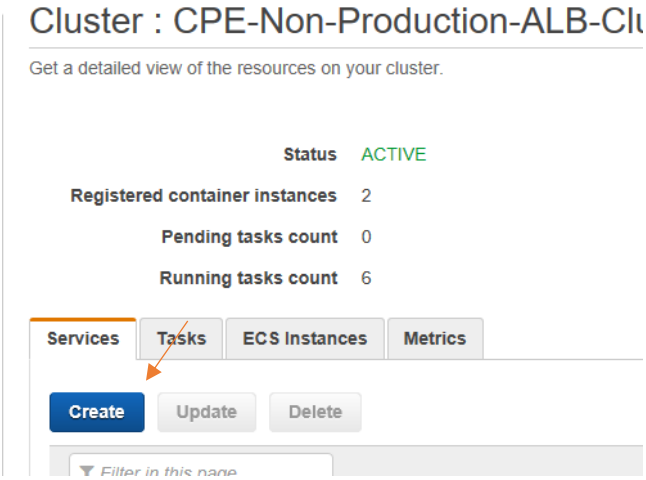
1. Open the AWS web ECS dashboard (us-east-1)

   https://console.aws.amazon.com/ecs/home?region=us-east-1

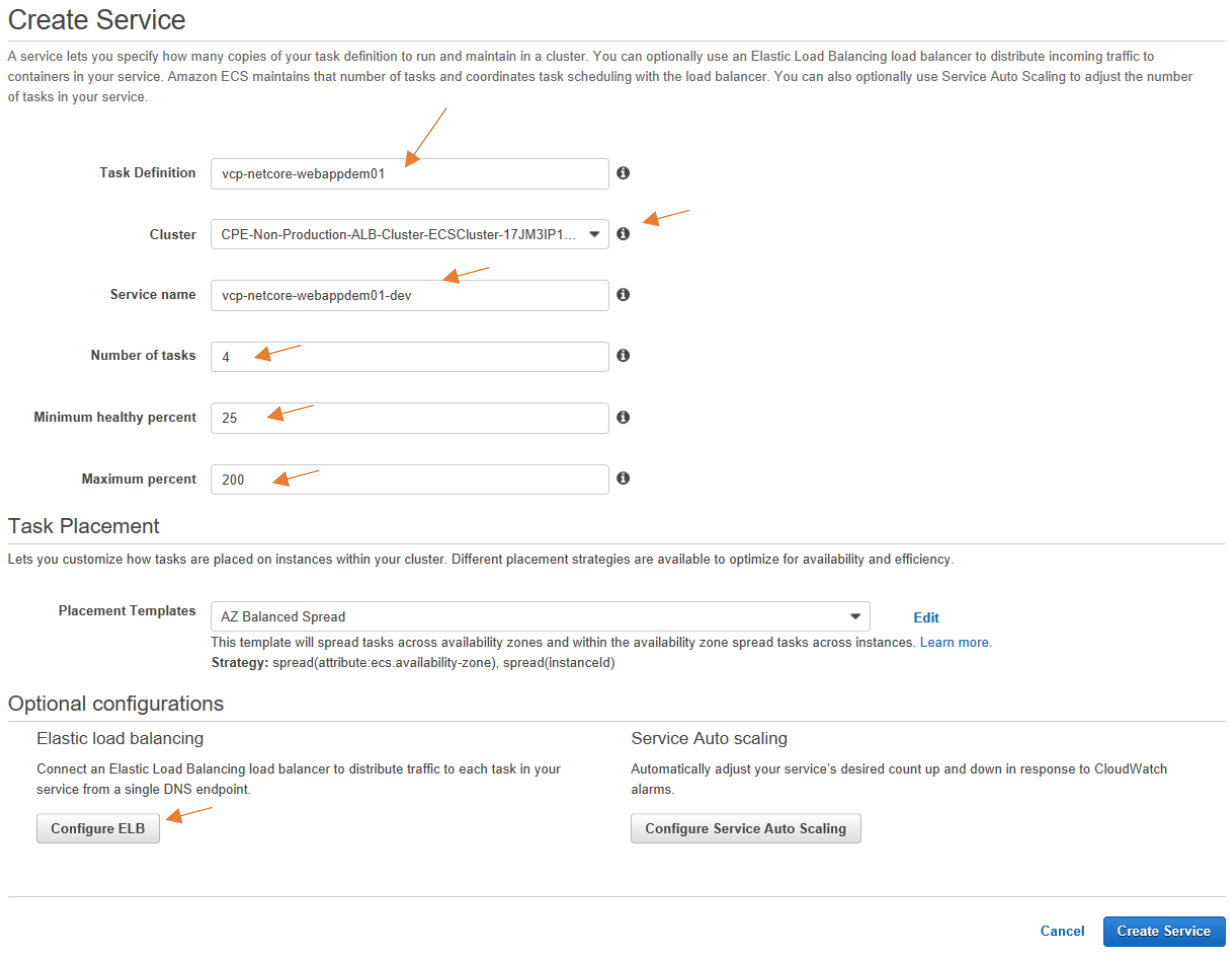
2. In the “Clusters” section open the ECS cluster you will use to create the service. In our example the cluster was already created by the MCS team and is called CPE-Non-Production…..QP7X (CPE = Content Platform Engineering)



3. In the “Services” tab click on “Create”

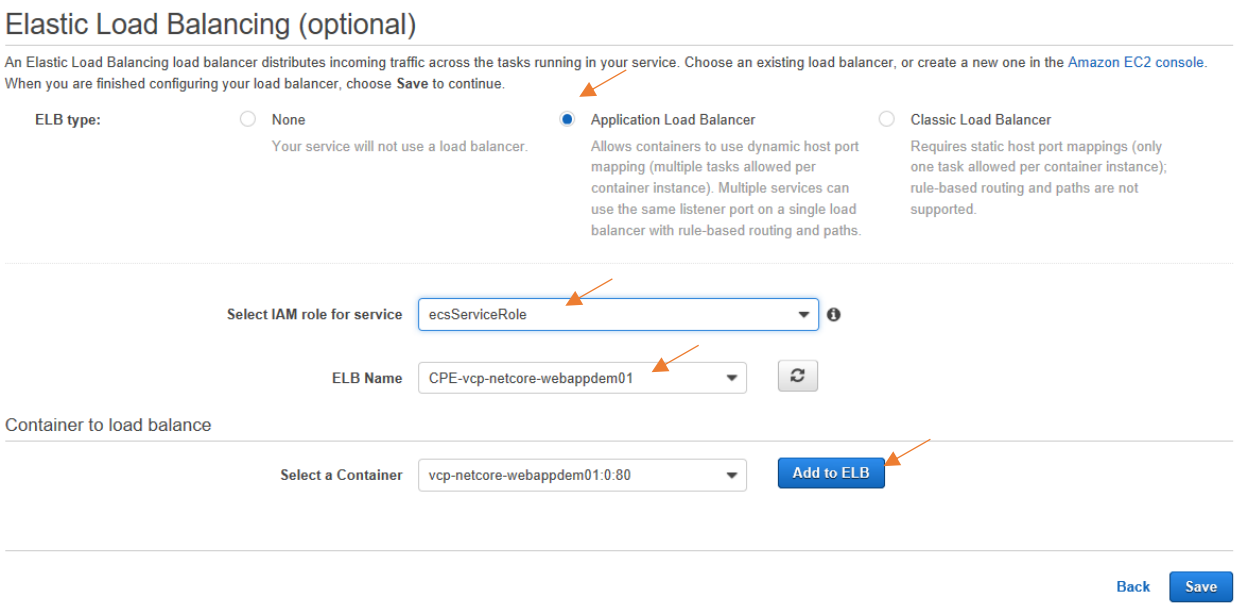


4. Fill the form as follow and click on “Configure ELB”



- **Task definition:** [repository name] (in this example is: “vcp-netcore-webappdem01”)
- **Cluster:** CPE-Non-Production…..QP7X
- **Service Name:** [repository name] (in this example “vcp-netcore-webappdem01-dev”, we added dev to remark that this service belongs to the DEV environment.)
- **Number of tasks:** as needed, more tasks means more reliability but more resource utilization. It should be defined by the “Lead engineer”
- **Minimum healthy percent:** 25 % (AWS will prevent killing task when the minimum healthy percent is reached. This is also used during deployments)
- **Maximum percent:** 200 % (During deployment, AWS can increase the number of tasks up to this percentage, this way AWS prevents down times of you service)

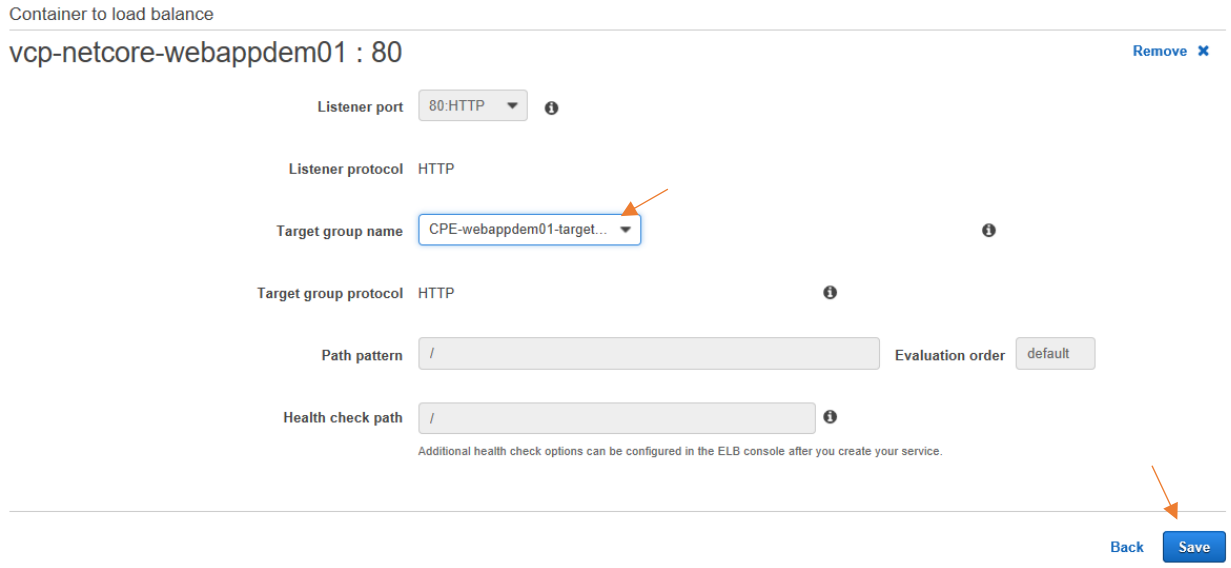
5. Fill the form and select the ELB (Elastic Load balancer) that we've just created before.



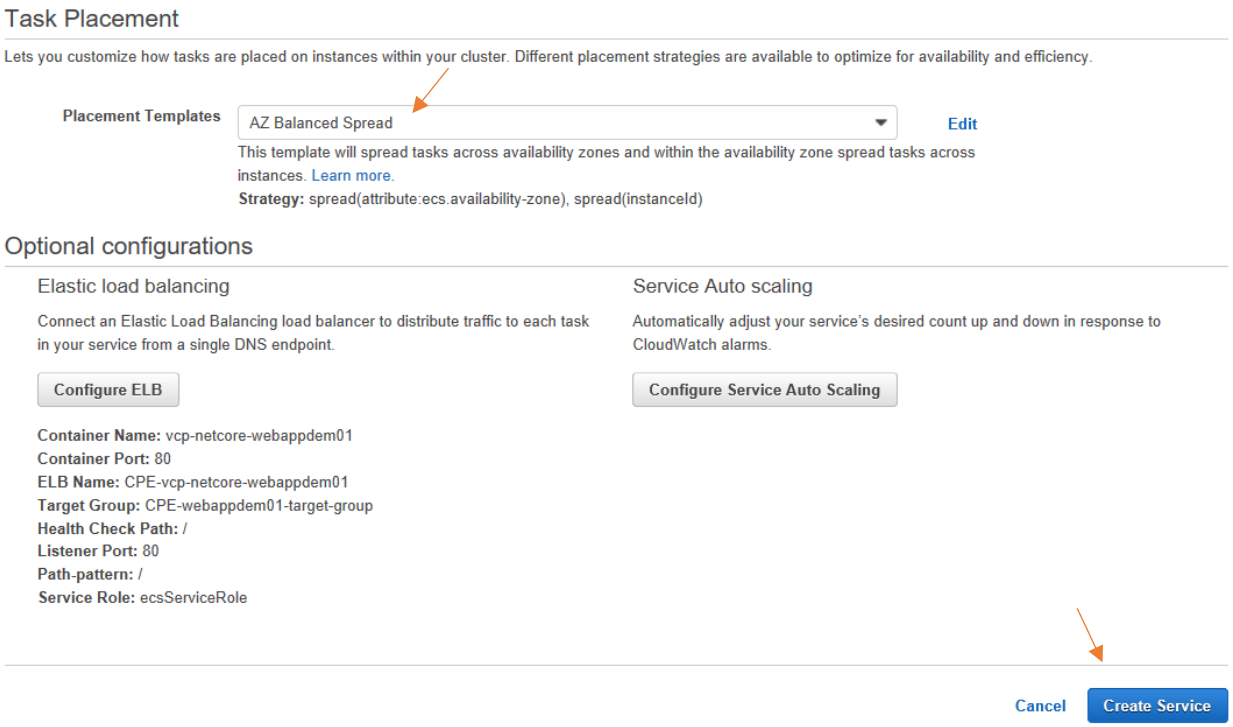
- **Select IAM role for service:** this information is provided by MCS. In our example we used "ecsServiceRole". This is a security configuration that allow the load balancer to operate.

  NOTE about terminology: ELB = AWS ALB = ALB = AWS Load Balancer = Load balancer.

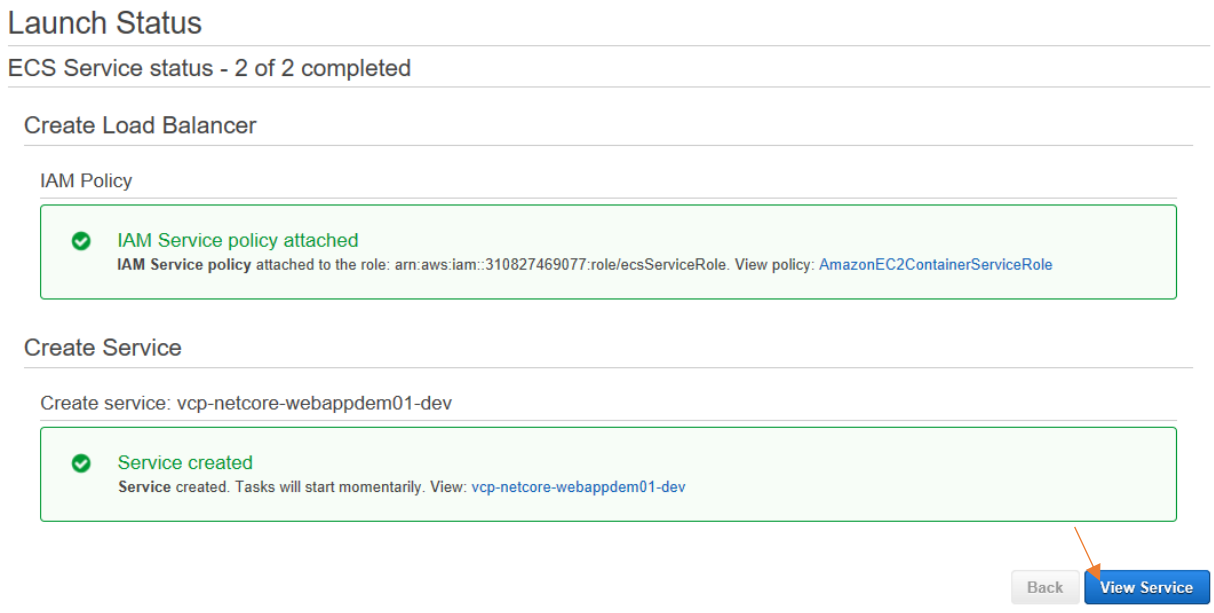
6. Then click on "Add to ELB", select the target group we've created before and click on "Save"



7. Make sure you task placement is “AZ Balanced Spread” and click on “Create service”.



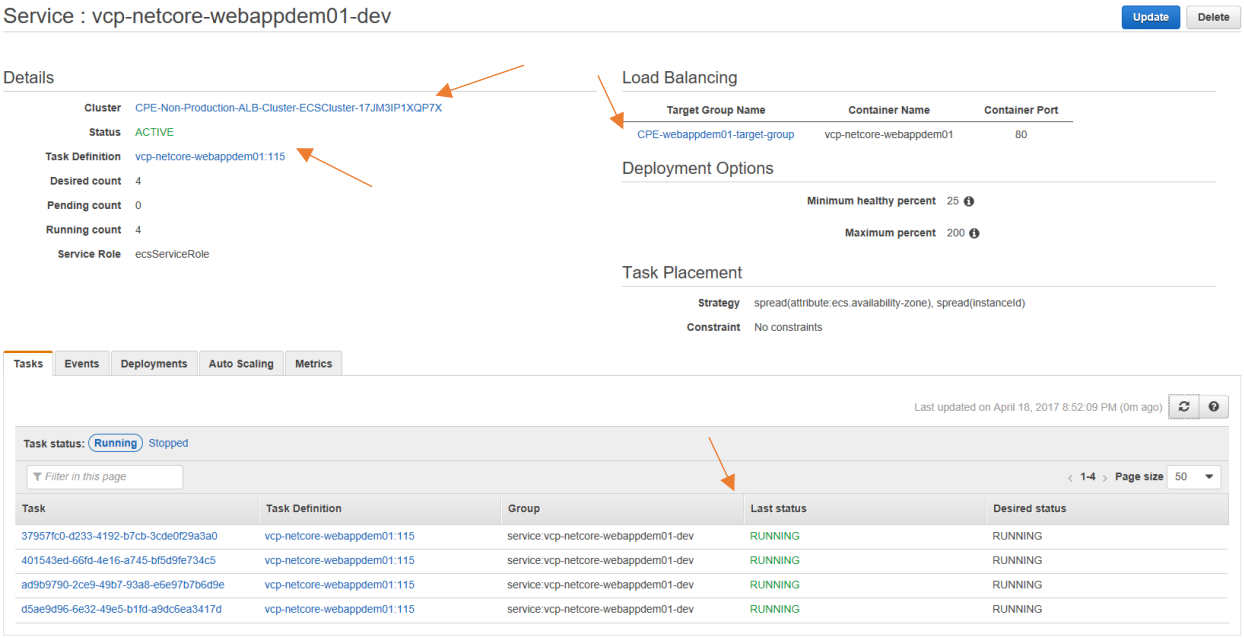
8. Done! after a couple of seconds you will see a confirmation page in green:



Click on “View Service” and go to next section

Final verification

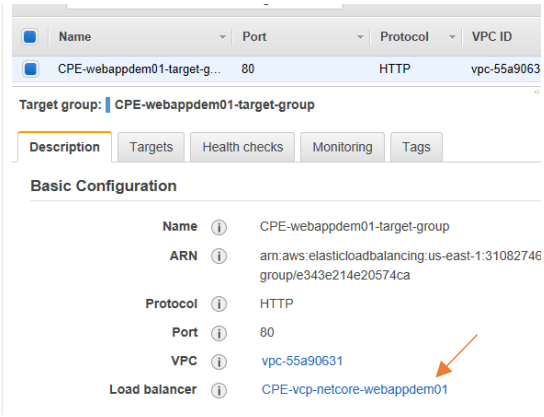
After a couple of minutes, you will see all your "Task" in RUNNING status. The service page shows the resources used: cluster, task definition and target group:



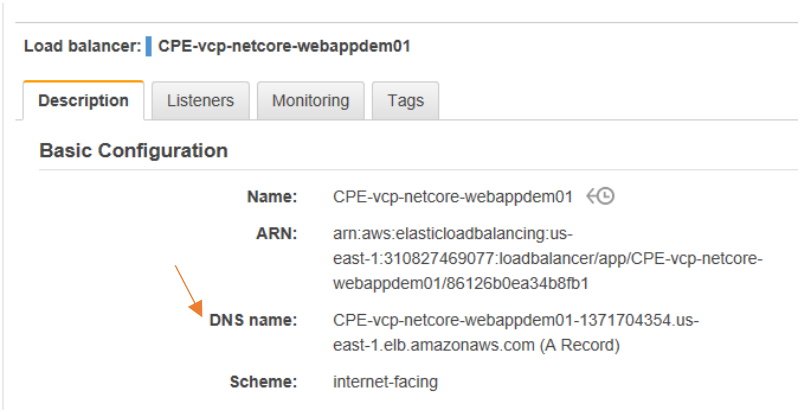
If everything is ok we should be able to open our POC web application in a browser, to do that we need to browse to the public DNS name of the AWS Load balancer:

1. Click on the "Target group"

2. Click on “Load balancer”

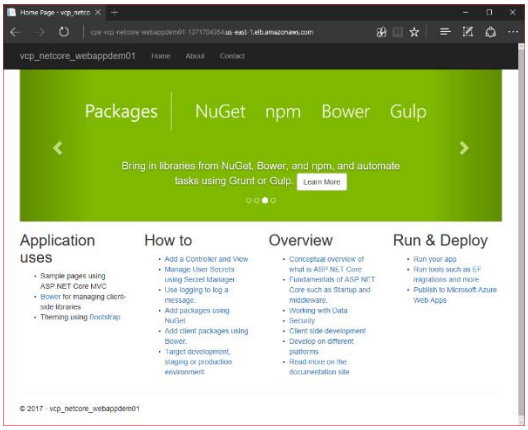


3. Copy the DNS name and use it as URL in your favorite browser:



In our case: http://cpe-vcp-netcore-webappdem01-1371704354.us-east-1.elb.amazonaws.com/

4. If you see your page all has worked out!



Congratulations! your system is hosted in AWS ECS.

So now the question is, will our new releases be deployed automatically in AWS? The answer is NO.

The next section will explain the details to ask AWS to deploy our new container “release-images” versions.

Setup execution environment in CI/CD pipeline

Introduction

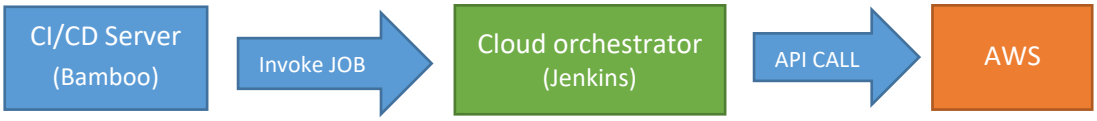
If you've followed this procedure in order, by now you have created three things:

- A continuous integration pipeline in Bamboo (CI)
- A continuous deployment pipeline in Bamboo (CD)
- The execution environment in AWS ECS.

There is problem, our CD pipeline just creates a new container "release-image" version and pushes it to the Viacom AWC ECR registry, but AWC ECS will not deploy this new version automatically. We have to commands AWS (thru its API) to request this update, we need something like "Hey AWS, please update all the 'Task' of my 'Service' with this new Docker image version". This section will explain how to update our CD pipeline to support this requirement.

Workflow overview

The MCS teams has implemented a "cloud orchestration layer" in front of AWS. This layer is a set of Jenkins jobs that you have to invoke when you want to interact with the AWS API. This means that our Bamboo CD pipeline will have to talk with Jenkins to update the AWS-ECS-Service-Tasks (or any other AWS resource) and for that we need to create a new "Task definition"; to simplify all the interaction we will use two Docker-based helper tools.



Pre-requirements

Here the requirements you need to implement the instructions of this section:

- **Jenkins token**: This is an authentication token provided by the MCS team that is needed to send job requests to the Jenkins API.
- **AWC ECS task definition template**: This is the task definition json template we created previously.
- **AWS ECS Service name.**
- **AWS ECS Cluster name.**
- **AWS Account name.**
- **Bitbucket repository to store the AWS resource definitions.**

Update AWS task definitions from CD pipeline

Every time that the CD pipeline runs a new VERSION number is created (build number and deploy number), this version number is used as tag for the container “release-image” hosted in AWS ECR. The objective here is to update the CD pipeline to create a new “Task definition” json file that includes that new VERSION-TAG.

The AWS resource definition repository

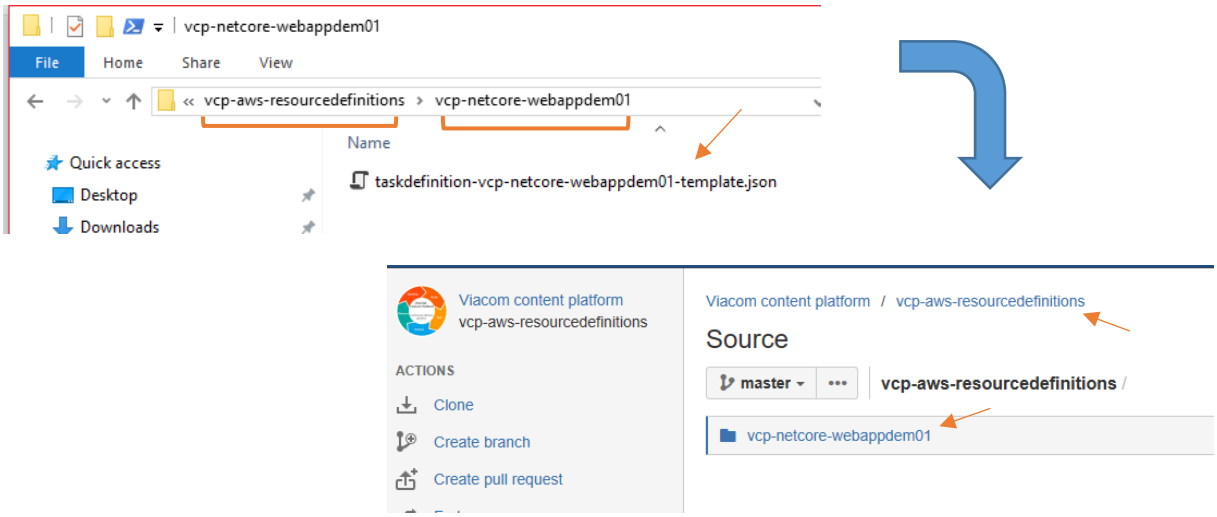
The new task definition json file will be stored in a repository in the Viacom Bitbucket instance called “vcp-aws-resourcedefinitions”. I will not explain how to do this because it is easy and because we’ve already explained how to create a repository in Bitbucket earlier. Anyway the repository has already been created and here the link:

https://stash.mtvi.com/projects/VCP/repos/vcp-aws-resourcedefinitions

NOTE

In our approach we will use the same aws-resource repository for all projects to reduce the number dependencies. But other approach could be having multiple “AWS resource definitions” repositories for every project.

You have to upload your “Task definition template” file to this repository under a subfolder named as you project repository name, for example:



To do this clone the repository locally, create the subfolder, copy the template json file, add changes, commit and push.

The containerized git-helper tool (VCP toolbox)

From Bamboo we are not allowed to updates any Bitbucket repository due to security restrictions, because of that we will have to create a custom Docker image with special SSH credentials and with the git client installed.

In the future we might need to create more engineering and development tools like this one. We will describe this family of tools as the "**Viacom content platform toolbox"** (VCP toolbox)

The container image has already been created and is available in this repository:

https://stash.mtvi.com/projects/VCP/repos/vcp-toolbox-docker-githelper

This is an agnostic tool that can be used for any Bamboo deploy project. In addition, it has been hosted in AWS ECR:

https://console.aws.amazon.com/ecs/home?region=us-east-1#/repositories/viacomcontentplatform:vcp-toolbox-docker-githelper

The procedure to create the git-helper container image and the instructions to publish to AWS ECR is explained in the document "VIACOM - Content Platform - Continuous Delivery Workflow - Toolbox - git-helper"

Update deploy project (part I)

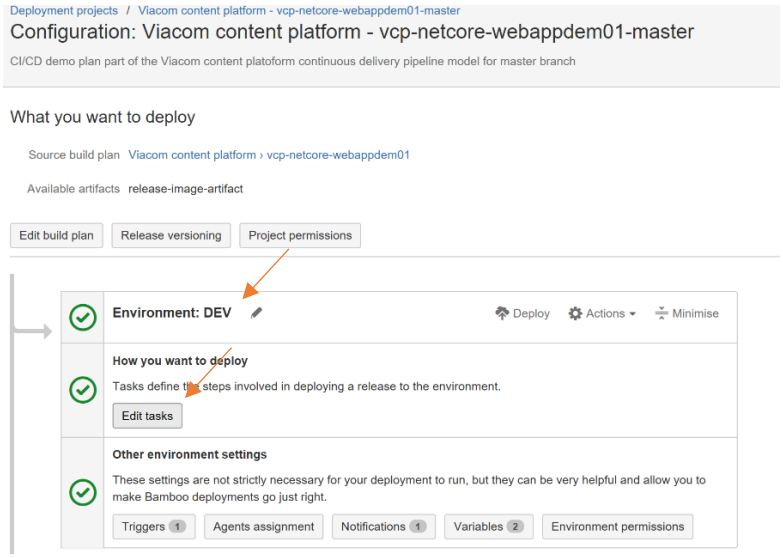
This is the final step that will enable a real continuous delivery workflow. The following steps will explain how to update our Bamboo deployment project for creating the "Task definition" json file on deploy-time and for updating the AWS execution environment by sending a request to Jenkins.

1. In Bamboo open the "vcp-netcore-webappdem01-master" deploy project and go to "Edit project"

   https://bamboo.vmn.io/deploy/viewDeploymentProjectEnvironments.action?id=69042178

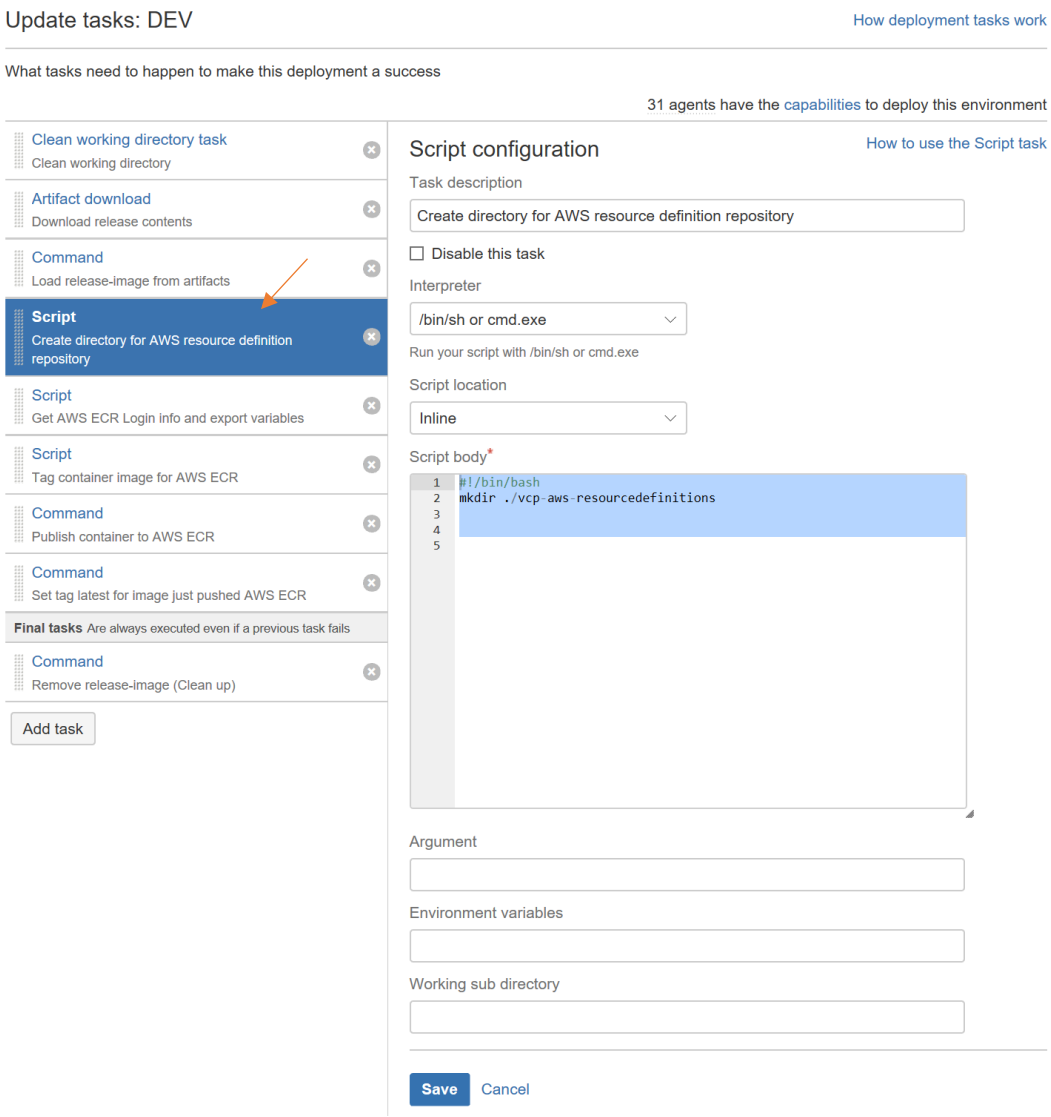
2. Click on "Edit task" of the DEV environment

   NOTE: The same steps have to be included into the QA, PROD environment and into the DEV environment of the development-branch (nightly build/deploy)



We will need to create more tasks in the correct position, follow the steps:

3. "Script / Create directory for AWS resource definition repository" (**Position #4**)



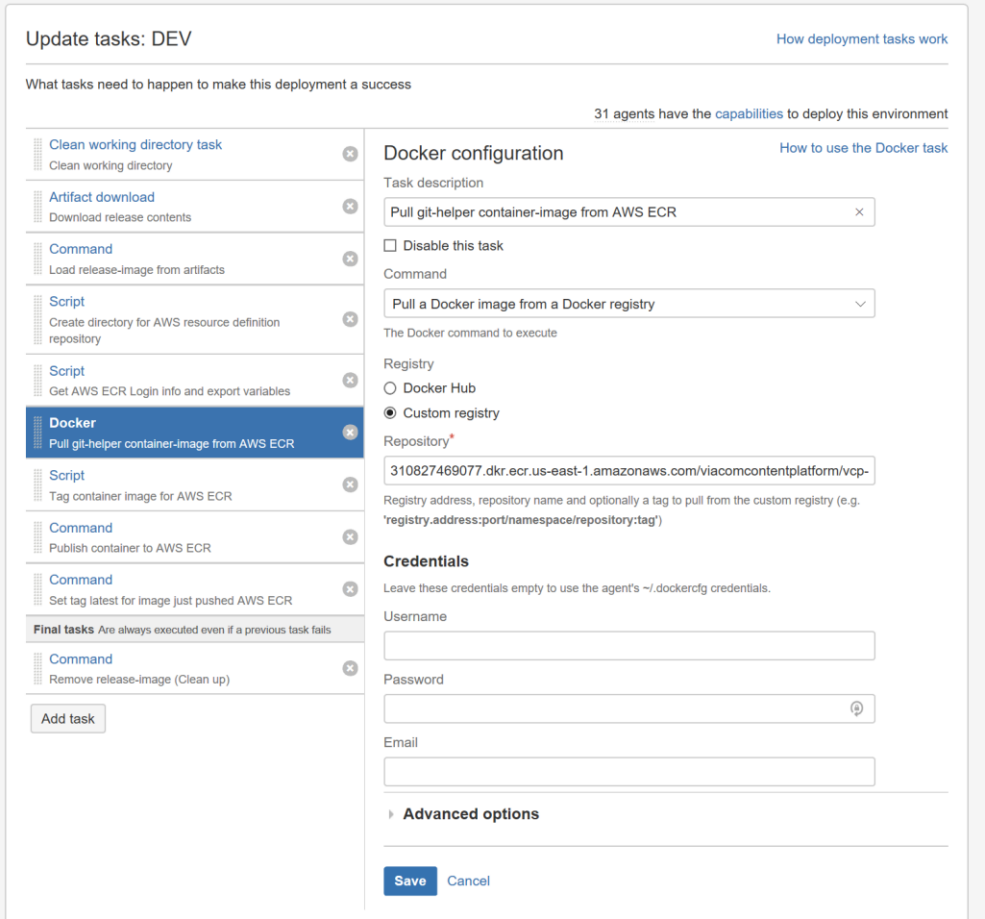
Script body

```
#!/bin/bash
mkdir ./vcp-aws-resourcedefinitions
```

Result

We create a temp folder in the bamboo agent that will be used to store the AWS-resource-definitions files pulled from the vcp-aws-resourcedefinitions bitbucket repository.

4. "Docker / Pull git-helper container-image from AWS ECR" (**Position #6**)

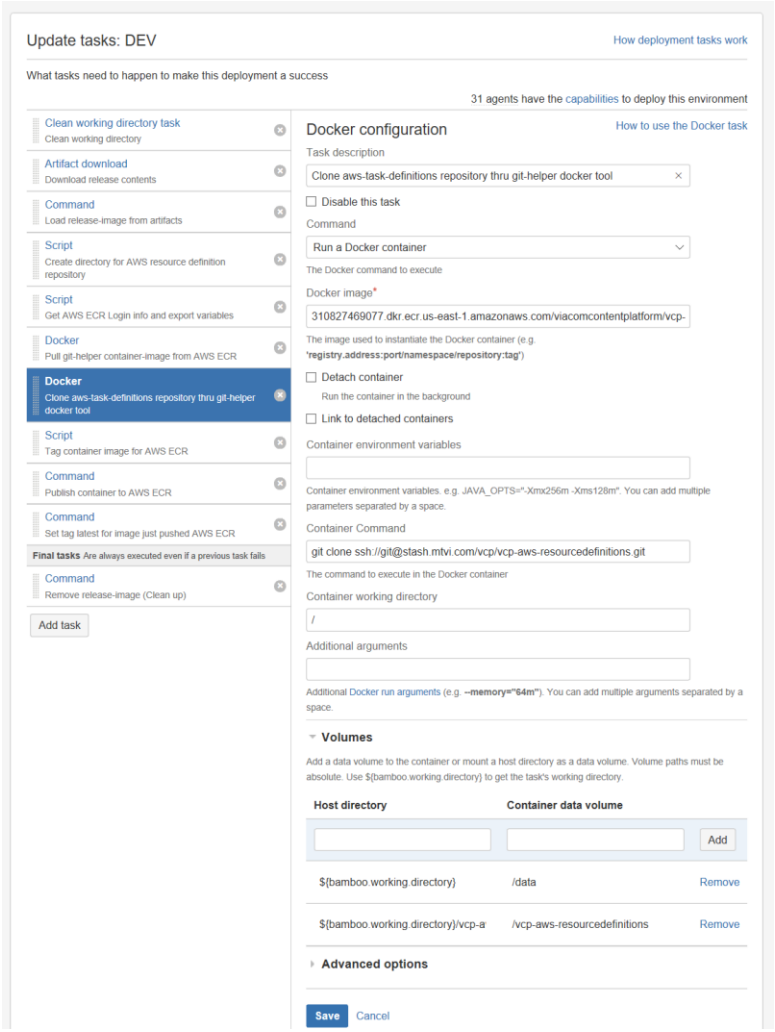


- **Command:** “Pull a Docker image from a docker registry”
- **Registry:** “Custom registry”
- **Repository:** 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-toolbox-docker-githelper:latest

Result

We are downloading the docker image of our git-helper tool that will allow us to clone the “vcp-aws-resourcedefinitions” Bitbucket repository and push back the new AWS Task definition.

5. "Docker / Clone aws-task-definitions repository thru git-helper docker tool" (**Position #7**)

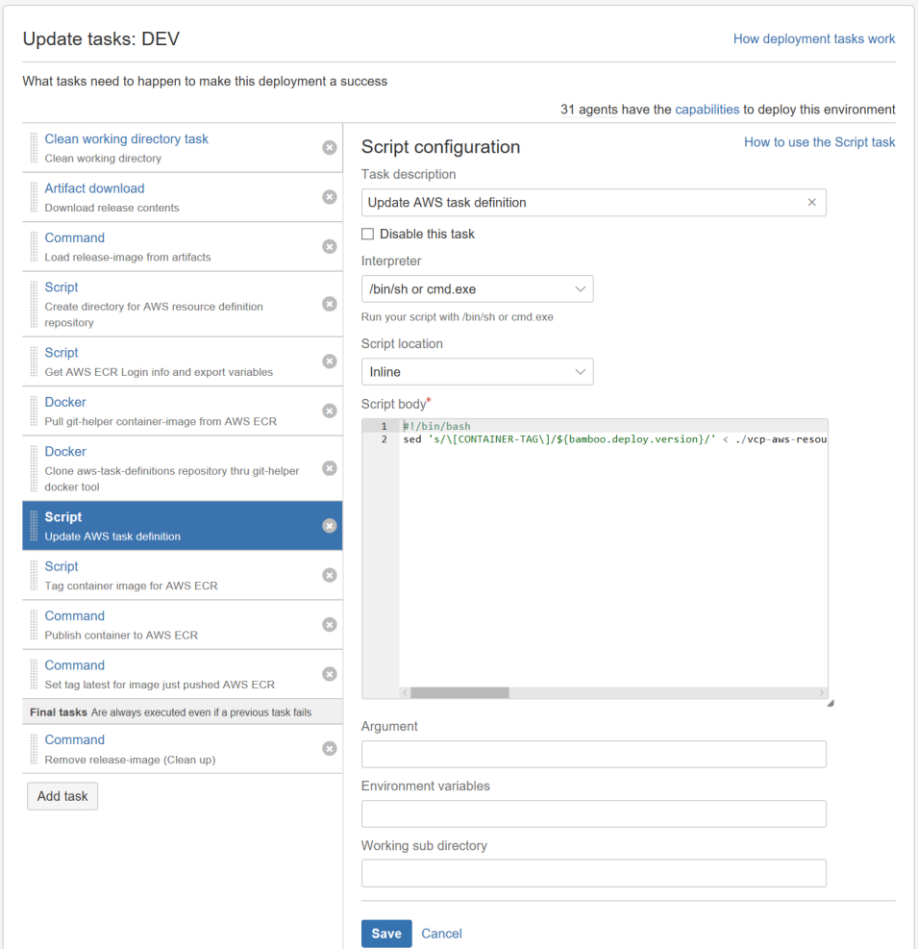


- **Command:** Run a Docker container
- **Docker image:** 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-toolbox-docker-githelper:latest
- **Container command:** `git clone ssh://git@stash.mtvi.com/vcp/vcp-aws-resourcedefinitions.git`
- **Container working directory: /**
- **Add volume**
    - Host: `${bamboo.working.directory}/vcp-aws-resourcedefinitions`
    - Container data volume: `/vcp-aws-resourcedefinitions`

Result

We clone the “vcp-aws-resourcedefinition” repository with our “git-helper” tool. The files are saved in a directory that is mounted to the underlying Bamboo agent file system, because of that the Bamboo agent we will be able to read the “AWS TASK template” json file.

6. "Script / Update AWS task definition" (**Position #8**)



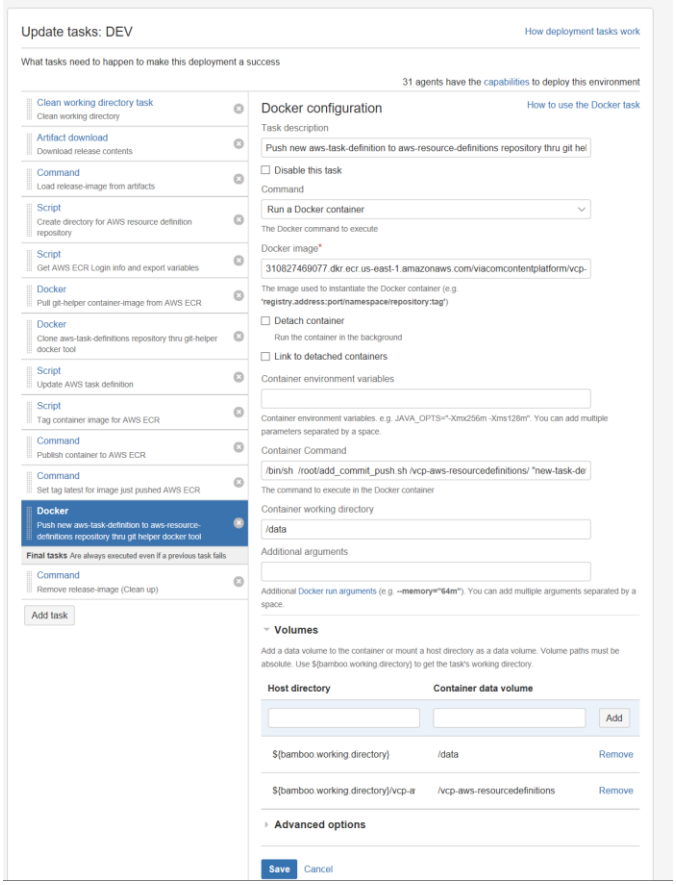
- **Interpreter:** /bin/sh or cmd.exe
- **Script location:** inline
- **Script body:**

```
#!/bin/bash
sed 's/\[CONTAINER-TAG\]/${bamboo.deploy.version}/' < ./vcp-aws-resourcedefinitions/vcp-netcore-webappdem01/taskdefinition-vcp-netcore-webappdem01-template.json >  ./vcp-aws-resourcedefinitions/vcp-netcore-webappdem01/taskdefinition-vcp-netcore-webappdem01-${bamboo.deploy.version}.json
```

Result

This script will read the “AWS Task definition” json template file of our “Service” (see the subfolder) and will replace the string token “[CONATINER-TAG]” with the VERSION number computed on deploy-time. The new JSON will be saved as a new “AWS Task definition” json file that will contain the VERSION number in its filename.

7. "Docker / Push new aws-task-definition to aws-resource-definitions repository thru git-helper docker tool" (**Position 12**)



- **Command:** Run a Docker container
- **Docker image:** 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-toolbox-docker-githelper:latest
- **Container command:**

  ```
  /bin/sh /root/add_commit_push.sh /vcp-aws-resourcedefinitions/ "new-task-definition-${bamboo.deploy.version}" ssh://git@stash.mtvi.com/vcp/vcp-aws-resourcedefinitions.git
  ```

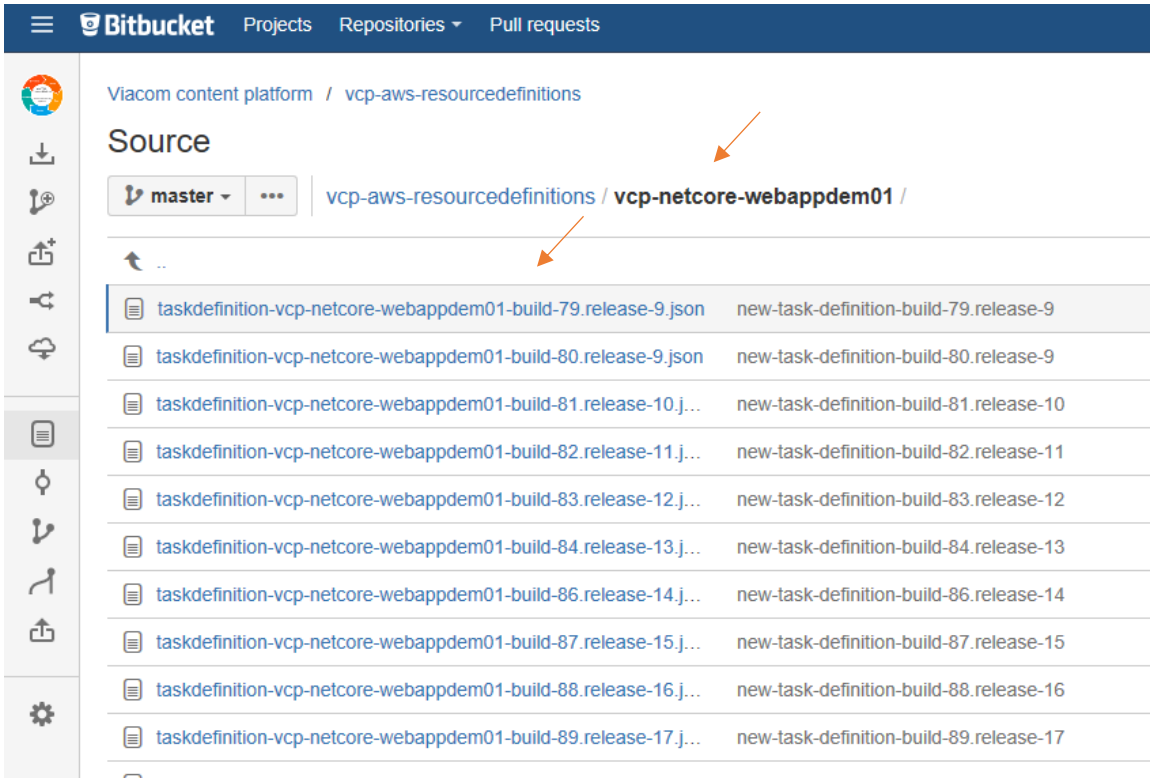
- **Container working directory:** /data
- **Add volume**
  - Host: `${bamboo.working.directory}/vcp-aws-resourcedefinitions`
  - Container data volume: `/vcp-aws-resourcedefinitions`

Result

We are using our git-helper tool to execute the script “add_commit_push.sh” that will push our new “AWS Task definition” to the vcp-aws-resourcedefinition repository in Bitbucket. (Note that the script needs 3 parameters: path, commit message and repository SSH URL)

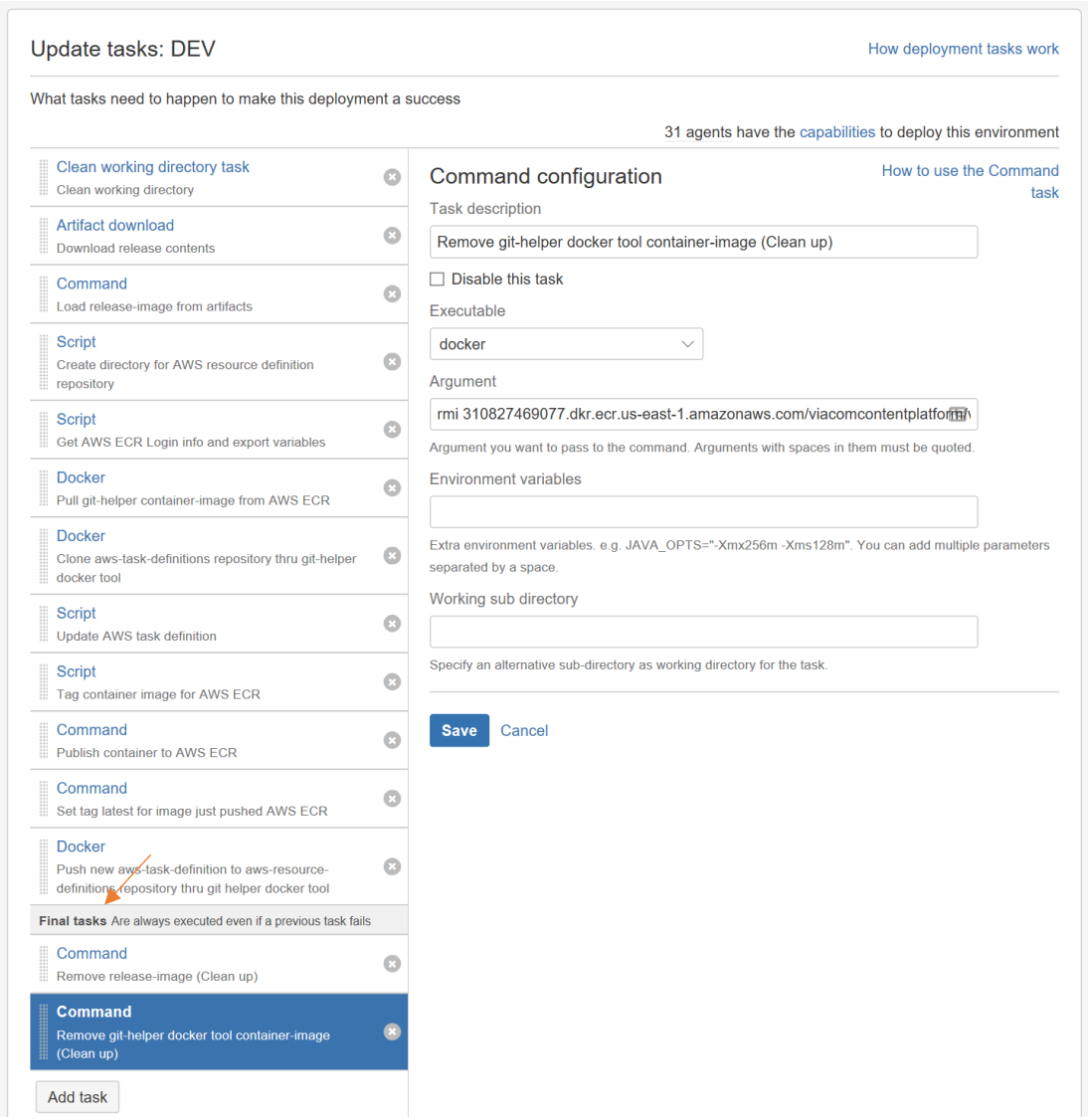
CONTEXT

Once the pipeline is up and running you will see that the “vcp-aws-resourcedefinition” repository will be populated as shown in the image below:



The image shows the “AWS task definitions” for every release/deploy that belongs the project “vcp-netcore-webappdem01” (subfolder)

8. "Command / Remove git-helper docker tool container-image (Clean up)" (**Last position of "Final task" section**)



- **Executable:** docker

- **Argument:**

```
rmi 310827469077.dkr.ecr.us-east-
1.amazonaws.com/viacomcontentplatform/vcp-toolbox-docker-githelper:latest
```

- **Result:**

We are deleting the git-helper docker image from the local docker registry of the bamboo agent.

What have we done so far and what's next?

We've updated the deploy pipeline to include the tasks that are responsible of creating the AWS Task definition file that will contain the new VERSION number and we hosted this new json file in the AWS resource definitions repository (vcp-aws-resourcedefinitions)

The last part of this CD pipeline is to say "Hey AWS, please stop all the Task instances running in our Service execution environment in AWS ECS, and starts new ones but using our brand-new release-container-image that we've just hosted in AWS ECR. ". We are going to do this by sending a request to Jenkins.

Jenkins is another CI/CD server, similar to Bamboo, that is used by the MCS to orchestrate the interaction with the Viacom AWS instance.

You send a request to Jenkins via HTTP POST and passing some arguments that defines the Jenkins-Job you want to execute (in our case it will be the Docker-ECS-Update), the location of the new "Task definition" json file, what is the Service you want to update, etc. Here an example in PowerShell:

```
$postParams = @{username='me';moredata='qwerty'}
$jenkinsJobUrl = "http://jenkins.us-east-1.aws.cloud.viacom.com:8080/buildByToken/buildWithParameters?"
$paramJob= "job=Docker-ECS-Update"
$paramToken= "&token=XXXXXXXX"
$paramRepo = "&stashRepo=stash.mtvi.com/scm/vcp/vcp-aws-resourcedefinitions.git"
$paramEmail = "&contactEmail=david.sorbona@vimn.com"
$paramAccount ="&RemoteAccountName=vmn-platform-nonprod"
$paramTaskDefinition = "&taskDef=vcp-netcore-webappdem01/taskdefinition-vcp-netcore-webappdem01-build-70.release-27.json"
$paramCluster = "&ECSCluster=aws:ecs:us-east-1:310827469077:cluster/CPE-Non-Production-ALB-Cluster-ECSCluster-
17JM3IP1XQP7X"
$paramService = "&ECSService=aws:ecs:us-east-1:310827469077:service/vcp-netcore-webappdem01"
$urlRequest = $jenkinsJobUrl + $paramJob + $paramToken + $paramRepo + $paramEmail + $paramAccount + $paramTaskDefinition
+ $paramCluster + $paramService

Invoke-WebRequest -Uri $urlRequest -Method POST
```

More information about the Jenkins job "Docker-ECS-Update":

https://confluence.mtvi.com/pages/viewpage.action?pageId=303532189

But in order to really know if the Jenkins job has completed or not you have to send another HTTP GET request to get its status. This is not so simple. Jenkins API returns an QUEUE NUMBER when you send the first request, eventually that entry will be populated with a "RUN NUMBER" and that number is the Id of your job, so you have to keep checking the Jenkins queue. Once you get the "RUN NUMBER" you have to check the JOB-RUN QUEUE until you get the final state (In progress, fail, success). To simplify this interaction we've created another docker helper tool called "vcp-toolbox-docker-jenkinshelper" that will execute all of this for you.

I won't explain the details of how the "Jenkins-helper" was created, but the process is pretty similar to the one we use to create the git-helper.

The repository of this tool is available here:
https://stash.mtvi.com/projects/VCP/repos/vcp-toolbox-docker-jenkinshelper/browse

It is also hosted in AWS ECR:
https://console.aws.amazon.com/ecs/home?region=us-east-1#/repositories/viacomcontentplatform:vcp-toolbox-docker-jenkinshelper

IMPORTANT

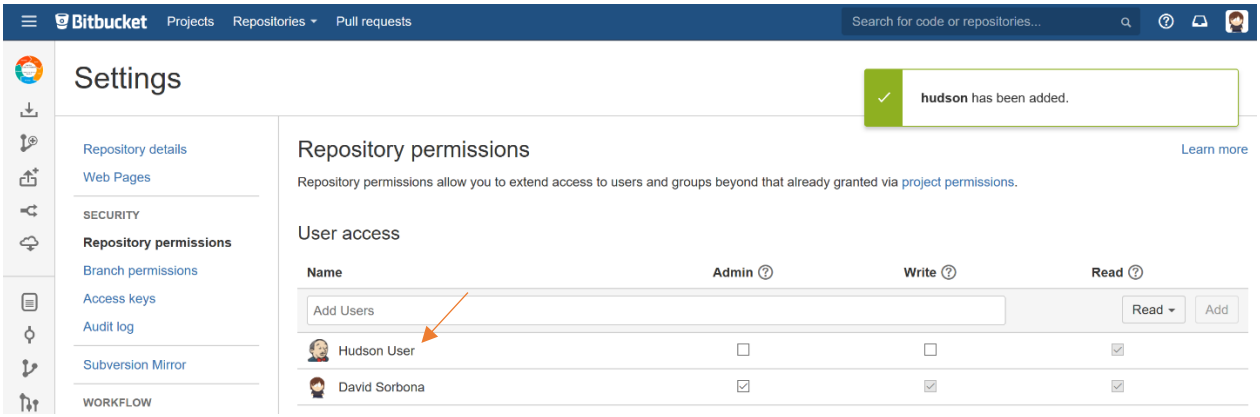
The deploy procedure for the production (PROD) AWS environment will demand the utilization of another Jenkins Job that introduces a manual step to approve the flight. We will cover this in a future version of this document.

Update deploy project (part II)

This section will explain how to update the Bamboo CD pipeline to send an “AWS ECS Service Update” requests to Jenkins via the Jenkins-helper tool.

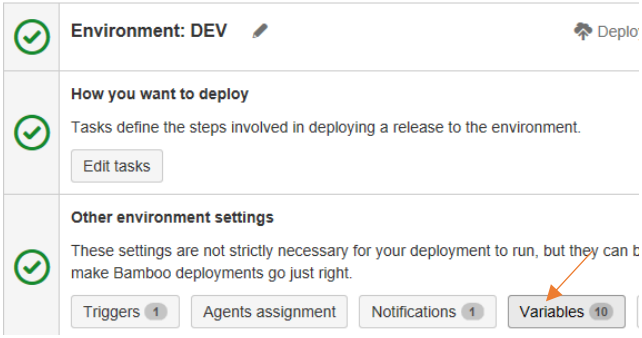
Pre-requirements

Jenkins needs to read the Bitbucket repository where we host the “AWS resource definition” files. Jenkins uses the account "Hudson User" for doing that. So in the “vcp-aws-resourcedefinitions” repository adds the account "Hudson User" with "Read" access.



Procedure

1. In Bamboo open the "vcp-netcore-webappdem01-master" deploy project, edit it and update the variables of the DEV environment:



https://bamboo.vmn.io/deploy/viewDeploymentProjectEnvironments.action?id=69042178

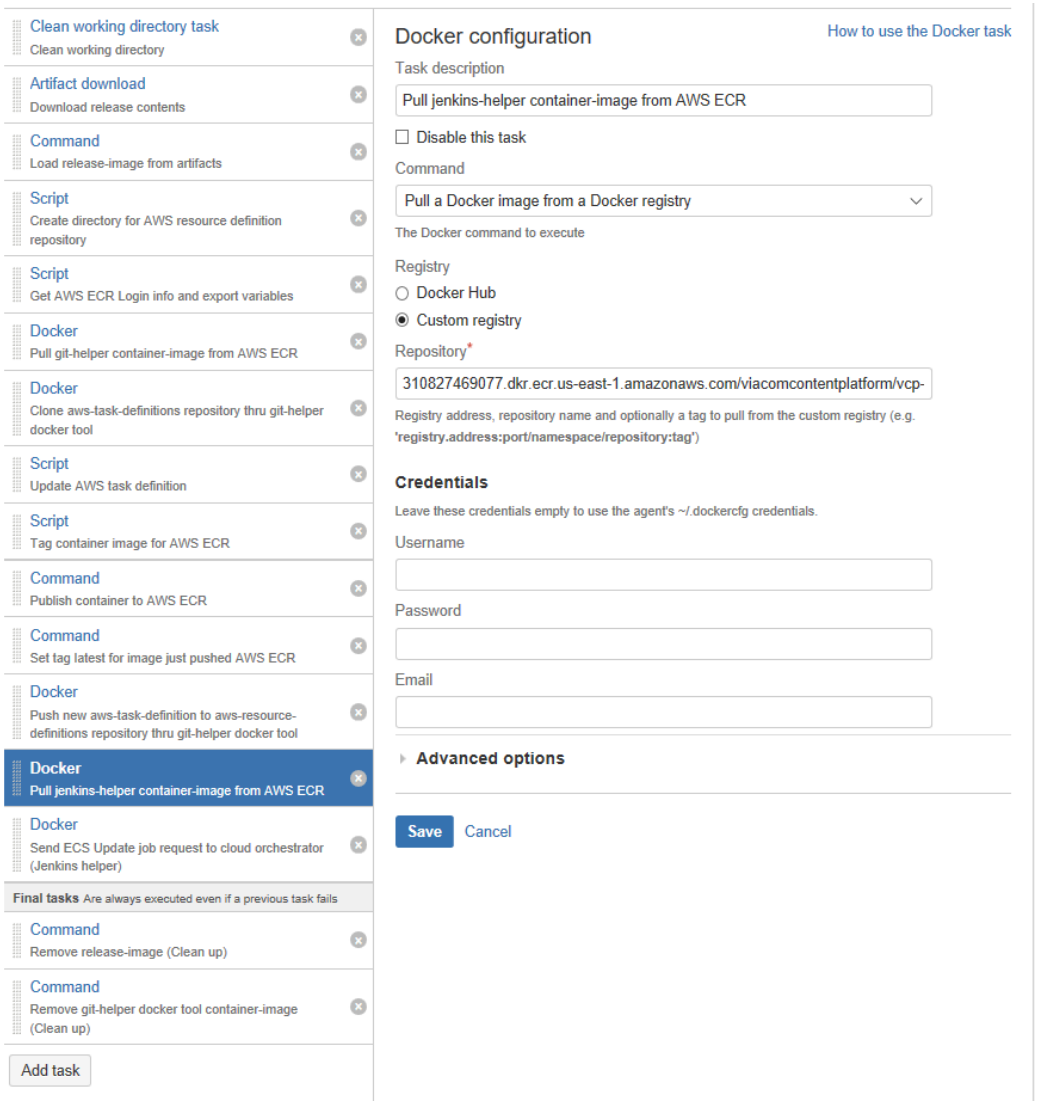
2. Add the following variables as shown in the image below:

Variable name	Value	
		Add
awsAccount	vmn-platform-nonprod	
awsAccountNumber	310827469077	
awsDefinitionsRepositoryName	vcp/vcp-aws-resourcedefinitions.git	
awsECSRole	BambooAccessFromCore	
awsEcsClusterName	CPE-Non-Production-ALB-Cluster-ECSCluster-17JM3IP1XQP7X	
awsEcsService	vcp-netcore-webappdem01-dev	
contactEmail	david.sorbona@vimn.com	
jenkinsJobName	Docker-ECS-Update	
jenkinsServerUrl	http://jenkins.us-east-1.aws.cloud.viacom.com:8080	
jenkinsTokenPassword	********	

Remember: The Jenkins token is provided by the MCS team.

Go back, click on “Edit tasks” and adds the following tasks in the CD pipeline:

3. “Docker / Pull jenkins-helper container-image from AWS ECR” **(Position 13)**



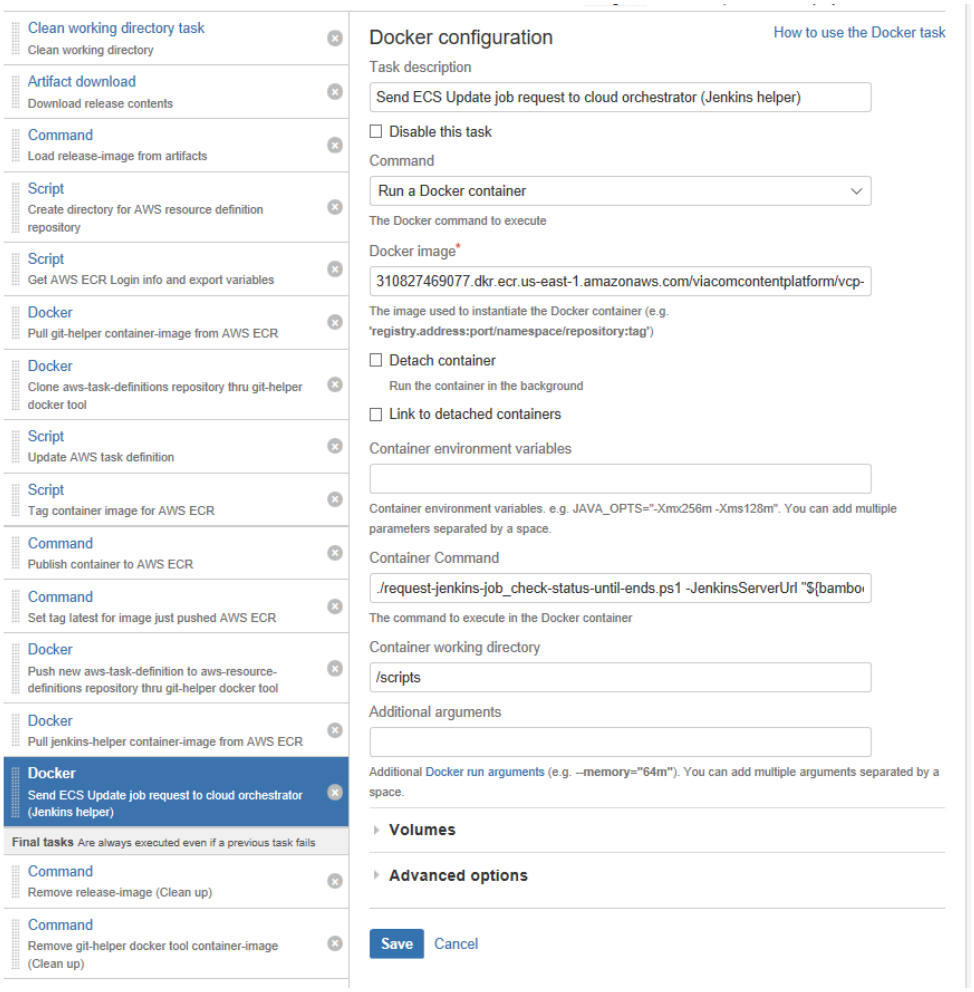
- **Command:** “Pull a Docker image from a docker registry”
- **Registry:** “Custom registry”
- **Repository:**

310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-toolbox-docker-jenkinshelper:latest

Result

We download the docker image of our Jenkins-helper tool that will allow us to command AWS to update the “Tasks” or our “Service” thru a Jenkins-job called “Docker-ECS-Update”.

4. “Docker / Send ECS Update job request to cloud orchestrator (Jenkins helper)” (**Position 14**)



- **Command:** Run a Docker container
- **Docker image:** 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-toolbox-docker-jenkinshelper:latest
- **Container command:**

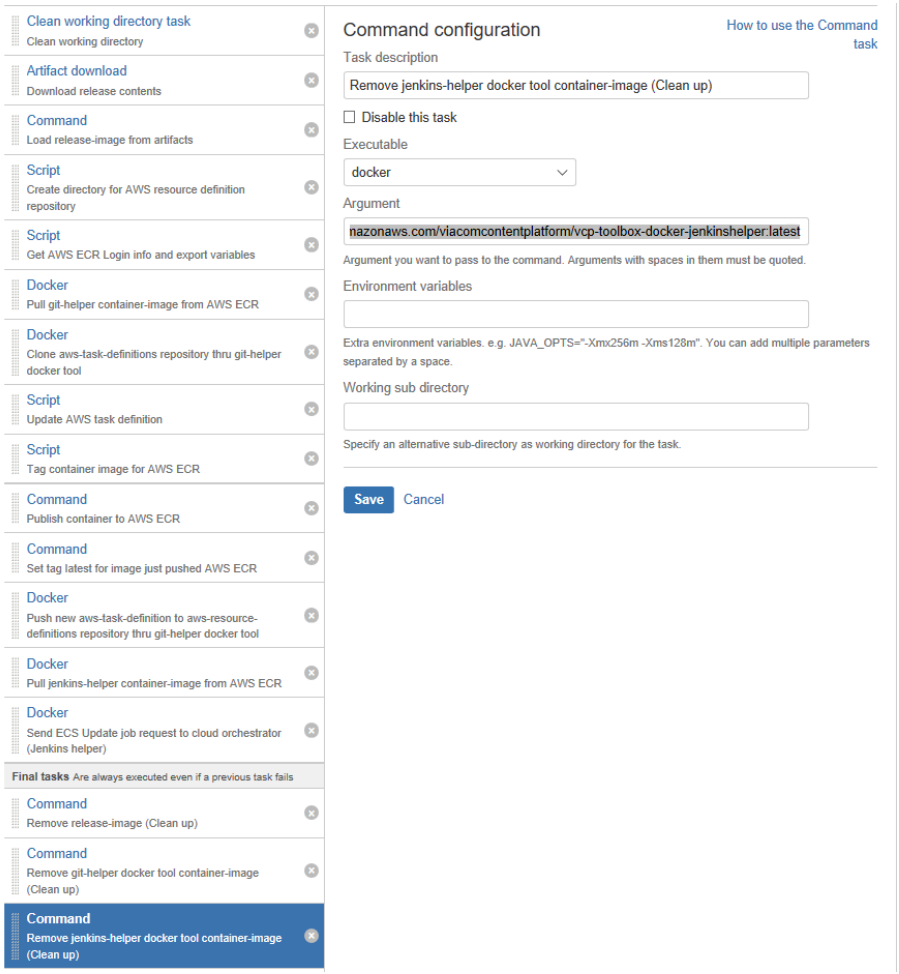
```
./request-jenkins-job_check-status-until-ends.ps1 -JenkinsServerUrl "${bamboo.jenkinsServerUrl}" -JenkinsJobName "${bamboo.jenkinsJobName}" -JenkinsToken "${bamboo.jenkinsTokenPassword}" -AwsDefinitionsRepositoryName "${bamboo.awsDefinitionsRepositoryName}" -ContactEmail "${bamboo.contactEmail}" -AwsAccount "${bamboo.awsAccount}" -AwsTaskDefinitionFileName "vcp-netcore-webappdem01/taskdefinition-vcp-netcore-webappdem01-${bamboo.deploy.version}.json" -AwsEcsClusterName "${bamboo.awsEcsClusterName}" -AwsEcsService "${bamboo.awsEcsService}"
```

- **Container working directory:** /scripts

Result

We are executing the PowerShell script “request-jenkins-job_check-status-until-ends.ps1” to send a “Docker-ECS-Update” request to Jenkins and to wait until the Jenkins-job completes.

5. “Command / Remove jenkins-helper docker tool container-image (Clean up)” (Last position of the “Final tasks” section)



- **Executable:** docker

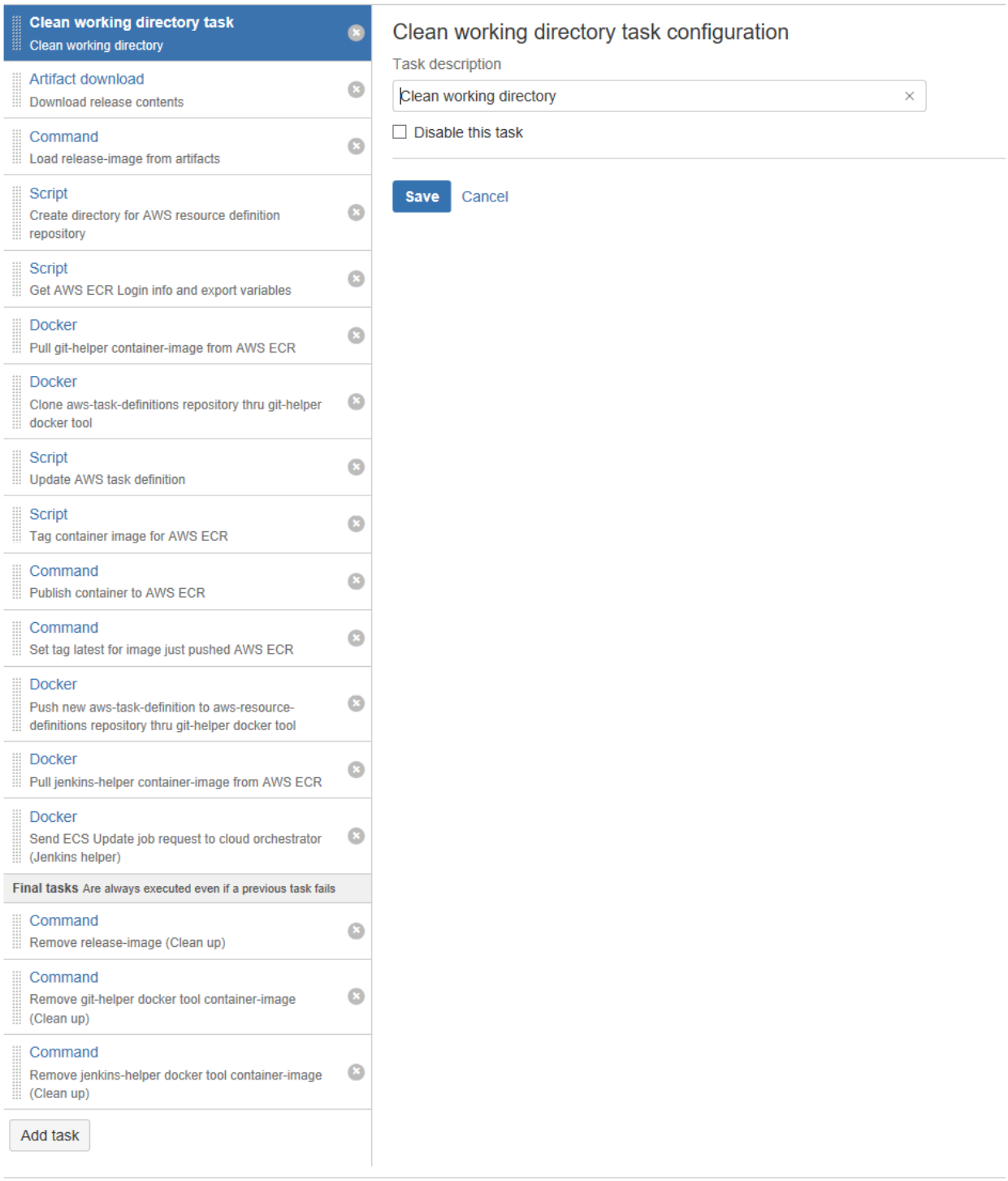
- **Argument:**

```
rmi 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-toolbox-docker-jenkinshelper:latest
```

Result

We are deleting the Jenkins-helper docker image from the local docker registry of the bamboo agent.

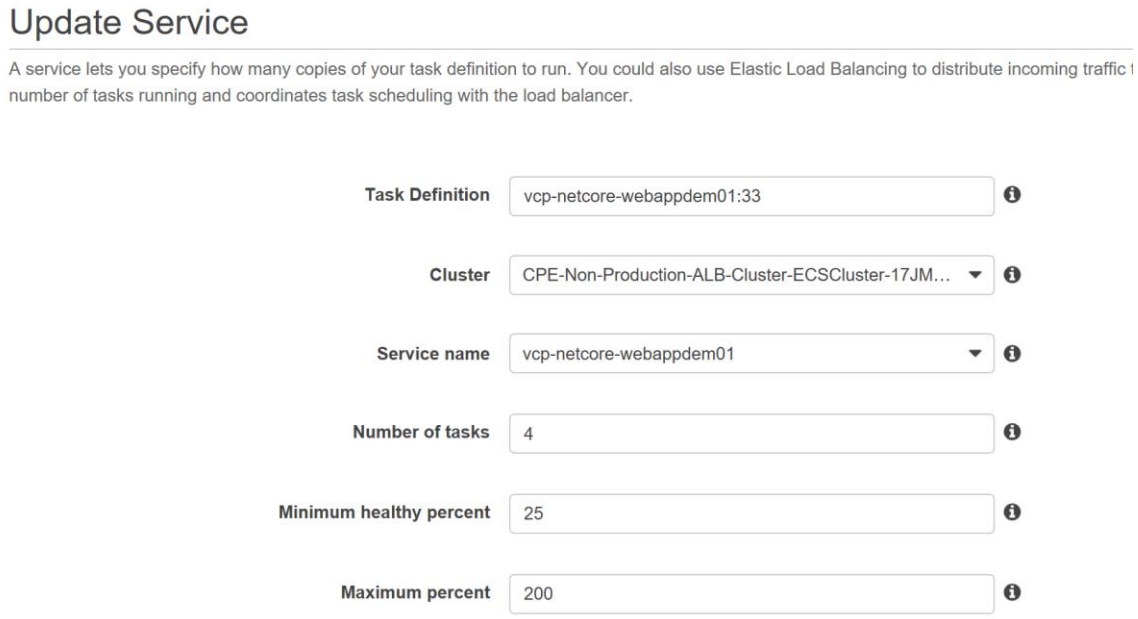
6. Done! The CD pipeline is ready. Here the full list of actions that your CD pipeline should have:



Congratulations your continuous delivery workflow is ready!

Additional information for AWS

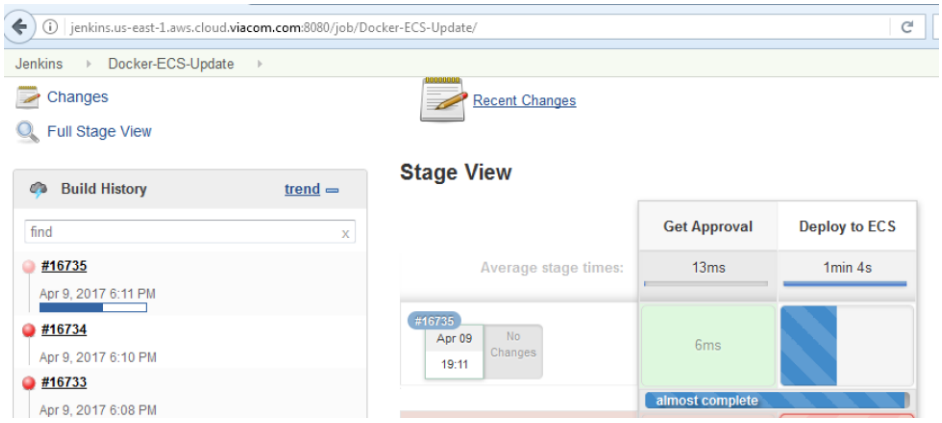
The AWS Service parameter "Number of tasks", "Minimum healthy percent" and "Maximum percent" defines the rules of how the Task can be replaced. Those values must be carefully set, if not the Jenkins ECS-Update job could not be able to complete successfully because you could reach an inconsistent state between the number of "healthy" task that must be running as minimum vs the "Stop Task" request that Jenkins is sending to AWS. In other words, if the number of task running are below the minimum limit, then Jenkins won't be able to stop that task and the Jenkins job could abort.



Additional information for Jenkins

The Jenkins dashboard URL is http://jenkins.us-east-1.aws.cloud.viacom.com:8080 . User your LDAP account. If you don't have access contact the "Tools and Engineering support" team.

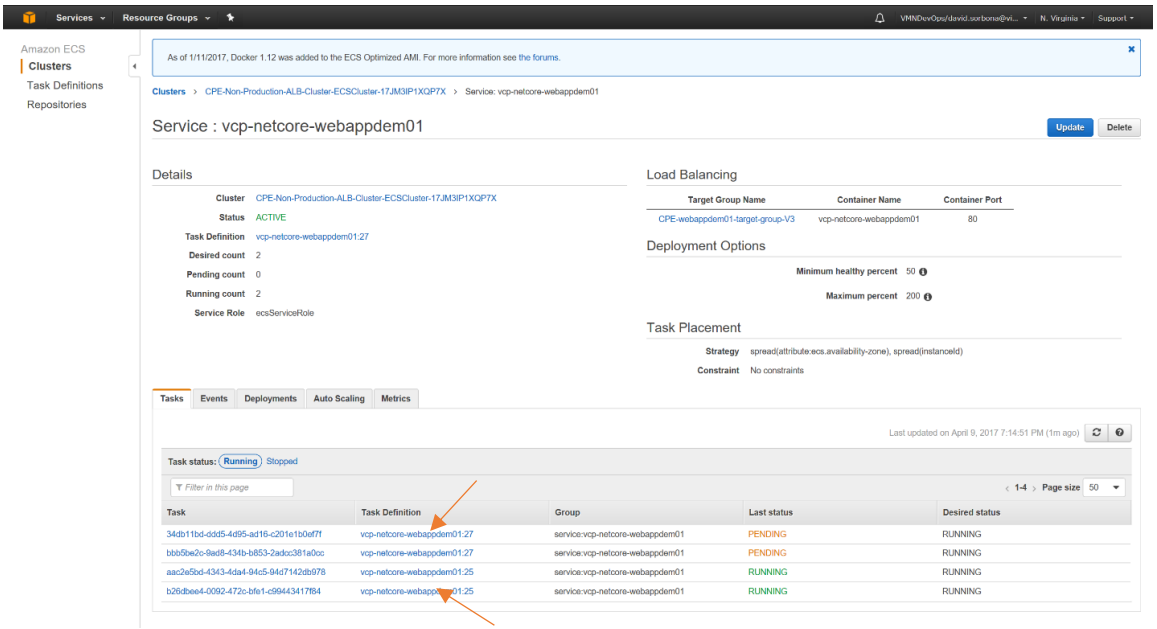
Every "Docker-ECS-Update" request will be queued and you could see its progress in the dashboard:



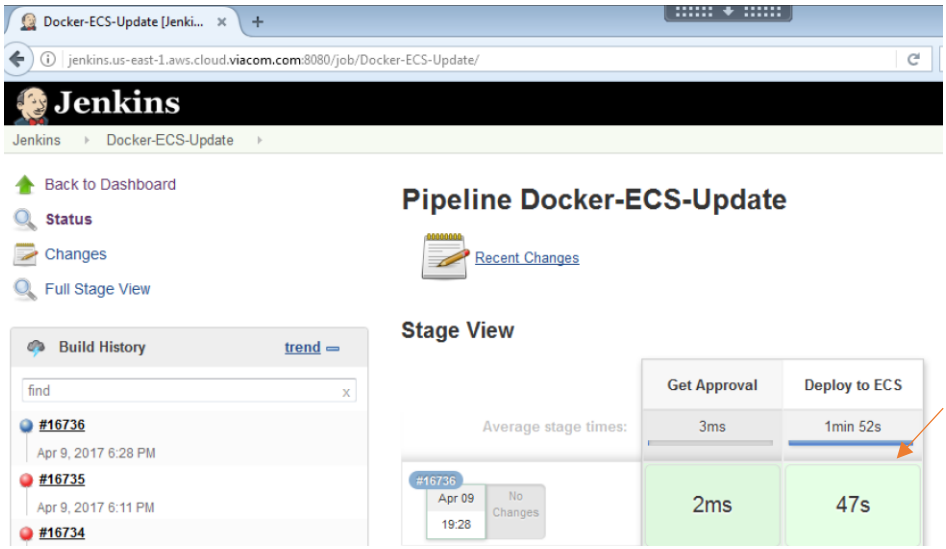
In addition, you can verify the status of the Jenkins-job from Bamboo deploy logs. This is possible because we are using the jenkins-helper tool. Here an example:

```
build   19-Apr-2017 06:13:38
build   19-Apr-2017 06:13:38    INFO: Jenkins job run (Docker-ECS-Update / 17635) status: In progress
build   19-Apr-2017 06:13:41    INFO: Checking job run #17635 status #151
build   19-Apr-2017 06:13:41
build   19-Apr-2017 06:13:41
build   19-Apr-2017 06:13:41
build   19-Apr-2017 06:13:41
build   19-Apr-2017 06:13:41
build   19-Apr-2017 06:13:41    INFO: Jenkins job run (Docker-ECS-Update / 17635) status: In progress
build   19-Apr-2017 06:13:44    INFO: Checking job run #17635 status #152
build   19-Apr-2017 06:13:44
build   19-Apr-2017 06:13:44
build   19-Apr-2017 06:13:44
build   19-Apr-2017 06:13:44
build   19-Apr-2017 06:13:44
build   19-Apr-2017 06:13:44    INFO: Jenkins job run (Docker-ECS-Update / 17635) status: In progress
build   19-Apr-2017 06:13:47    INFO: Checking job run #17635 status #153
build   19-Apr-2017 06:13:47
build   19-Apr-2017 06:13:47
build   19-Apr-2017 06:13:47
build   19-Apr-2017 06:13:47
build   19-Apr-2017 06:13:47
build   19-Apr-2017 06:13:47    INFO: Jenkins job run (Docker-ECS-Update / 17635) status: In progress
build   19-Apr-2017 06:13:50    INFO: Checking job run #17635 status #154
build   19-Apr-2017 06:13:50
build   19-Apr-2017 06:13:50
build   19-Apr-2017 06:13:50
build   19-Apr-2017 06:13:50
build   19-Apr-2017 06:13:50
build   19-Apr-2017 06:13:50    INFO: Jenkins job run status: SUCCESS
build   19-Apr-2017 06:13:53    SUCCESS: Jenkins job (Docker-ECS-Update / 17635) completed OK
gimpl   19-Apr-2017 06:13:53    Finished task 'Send ECS Update job request to cloud orchestrator (Jenkins helper)'
```

Jenkins will talk with AWS to remove current task instances of the service; then will update the Task definition so the service can start new task instances using the new container "release-image" hosted in the Viacom AWS ECR instance. The following image shows two tasks using the old version of the task definition (v25) and two tasks using the new version 27:



Once all AWS Tasks instances are updated, Jenkins will report that our “Docker-ECS-Update” request has completed with success.



Wrapping up

Final words

The document has explained how to create an end-to-end agnostic and container-oriented continuous delivery workflow for the “Viacom Content Platform initiative” based on Bitbucket, Bamboo for CI/CD, Jenkins for CD-AWS orchestration, AWS ECR for container image repository and AWS ECS as execution environment.

Roadmap

v1.4: New section: Adaptive changes for NPM workloads

v1.5: New section: Adaptive changes for Java workloads

v1.6: How to work with CloudFormation templates.

v1.7: New section: Adaptive changes for AWS PROD environment.

Glossary

Git	Distributed version control system for tracking changes in computer files and coordinating work on those files among multiple people.
Bamboo	Continuous integration server
Bitbucket	Web-based hosting service for projects
Stash	Former name of Bitbucket
Changeset	A “changeset” is an atomic collection of changes to files in a repository. It contains all recorded local modification that lead to a new revision of the repository. https://en.wikipedia.org/wiki/Changeset
Deployment unit	Any facet of an application component at run time or deployment time. A “Deployment Unit” is sufficiently complete to be downloadable, and run on a node
Development branch	Integration branch used during branch merging.
Feature branch	Used to develop new features for the upcoming or a distant future release.
Working directory,	It’s useful to think of the working directory as "the changeset I’m about to commit". If the working directory has no local modifications it is said to be *clean.*
AWS VCP	Amazon Virtual Private Cloud (Amazon VPC) lets you provision a logically isolated section of the Amazon Web Services (AWS) cloud where you can launch AWS resources in a virtual network that you define.
AWS Task definition	Amazon ECS task definitions is an abstraction that defines a set of properties of the “component” of your application. This is a text file in JSON format that describes one or more containers that form your application.

Useful links

What is Amazon EC2 Container Service?

http://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html