

# 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 .....               | 6  |
| Authors.....                                   | 6  |
| Collaborators.....                             | 6  |
| Change log.....                                | 6  |
| Abstract.....                                  | 7  |
| Scope.....                                     | 7  |
| Objective .....                                | 8  |
| Requirements.....                              | 8  |
| Continuous delivery pipeline .....             | 9  |
| Overview .....                                 | 9  |
| Requirements and restrictions.....             | 10 |
| Out of scope in this version .....             | 11 |
| Development workflow .....                     | 12 |
| General procedure .....                        | 12 |
| Development and integration strategies .....   | 13 |
| Personal/experimental branch strategy .....    | 13 |
| Pull requests strategy.....                    | 14 |
| Good practices to avoid merge conflicts .....  | 16 |
| Code quality attributes .....                  | 17 |
| Source code management .....                   | 18 |
| Introduction .....                             | 18 |
| Code repository in server.....                 | 18 |
| Repository setup procedure .....               | 19 |
| Code repository setup in development unit..... | 20 |

|  |    |
|--|----|
| Committing and pushing changes.....  | 21 |
| The code, track, commit and push loop.....   | 22 |
| Branch management.....   | 23 |
| Continuous integration pipeline .....  | 25 |
| Introduction .....   | 25 |
| Build plan setup .....   | 26 |
| Configure tasks.....   | 28 |
| Why do we use different images for building and for release?.....                  | 40 |
| Additional configurations.....   | 40 |
| Plan configuration.....  | 40 |
| Job configuration .....  | 43 |
| Branch management.....   | 45 |
| Setup branches .....   | 45 |
| Setup branch plans for automatic integration strategy.....                         | 47 |
| Setup automatic plan branch generation for feature branches .....                  | 48 |
| Setup predefined plan branch for “development branch”.....                         | 50 |
| Setup predefined plan branch for “development branch with master integration”..... | 51 |
| Additional observations for non-executable artifacts.....                          | 53 |
| Container registry .....   | 54 |
| Introduction .....   | 54 |
| Pre-requirements.....  | 54 |
| AWS ECR setup.....   | 54 |
| Continuous deployment pipeline.....  | 57 |
| Introduction .....   | 57 |
| Pre-requirements.....  | 57 |

|  |    |
|--|----|
| Deployment project setup for “master branch” .....     | 57 |
| Deployment project tasks setup .....                   | 60 |
| Finishing environments setup.....                      | 66 |
| Deployment triggers .....                              | 66 |
| Deployment notifications .....                         | 67 |
| Environment variables .....                            | 67 |
| Create QA and PROD environments. ....                  | 68 |
| Set project permissions.....                           | 69 |
| Deployment project setup for “development branch”..... | 70 |
| Working with deployments.....                          | 72 |
| Flag deployment as approved or broken .....            | 72 |
| Rollback deployment .....                              | 73 |
| Manual deployment.....                                 | 74 |
| Setup execution environment in cloud provider .....    | 75 |
| Introduction .....                                     | 75 |
| Architecture overview.....                             | 75 |
| AWS resources and governance .....                     | 76 |
| Environments: prod vs. non-prod .....                  | 76 |
| Pre-requirements.....                                  | 77 |
| Create task definition.....                            | 77 |
| Converting the task definition into a template.....    | 81 |
| Setup load balancer, target and security groups .....  | 82 |
| Setup service .....                                    | 87 |
| Final verification.....                                | 91 |
| Setup execution environment in CI/CD pipeline .....    | 93 |

|   |     |
|---|-----|
| Introduction .....                                    | 93  |
| Workflow overview .....                               | 93  |
| Pre-requirements .....                                | 93  |
| Update AWS task definitions from CD pipeline .....    | 94  |
| The AWS resource definition repository .....          | 94  |
| The containerized git-helper tool (VCP toolbox) ..... | 95  |
| Update deploy project (part I) .....                  | 95  |
| What have we done so far and what's next? .....       | 103 |
| Update deploy project (part II) .....                 | 104 |
| Pre-requirements .....                                | 104 |
| Procedure .....                                       | 104 |
| Additional information for AWS .....                  | 110 |
| Additional information for Jenkins .....              | 110 |
| Wrapping up .....                                     | 113 |
| Final words .....                                     | 113 |
| Roadmap .....   | 113 |
| Glossary .....  | 114 |
| Useful links .....                                    | 115 |

## 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<br><a href="https://stash.mtv.com">https://stash.mtv.com</a>   |
| Continuous integration + testing + building + package | Bamboo<br><a href="https://bamboo.vmn.io">https://bamboo.vmn.io</a>  |
| Container technology                                  | Docker   |
| Continuous deployment                                 | Bamboo<br><a href="https://bamboo.vmn.io">https://bamboo.vmn.io</a>  |
| Container images repository                           | AWS Elastic Container Repository (ECR)   |
| Cloud orchestration layer                             | Jenkins<br><a href="http://jenkins.us-east-1.aws.cloud.viacom.com:8080">http://jenkins.us-east-1.aws.cloud.viacom.com:8080</a> |
| 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.mtv.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.



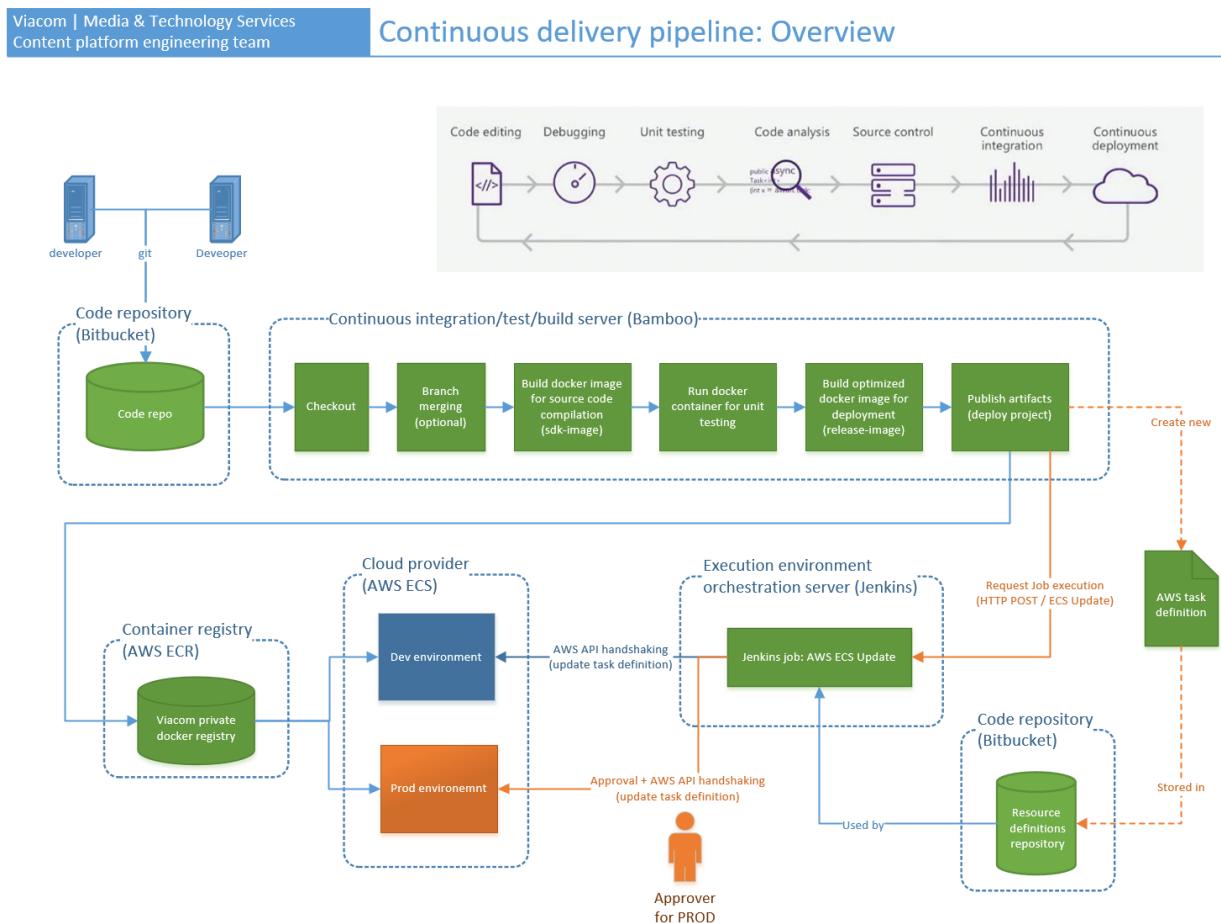
## Continuous delivery pipeline

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)
- Your 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.

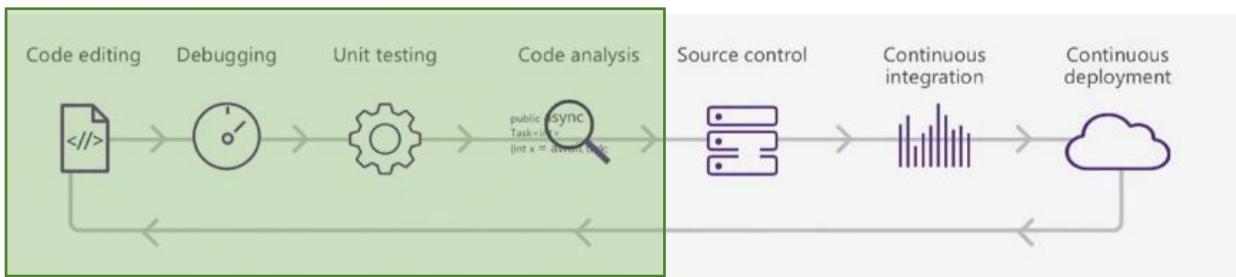


## Out of scope in this version

- 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 changes and your 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 → master**

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

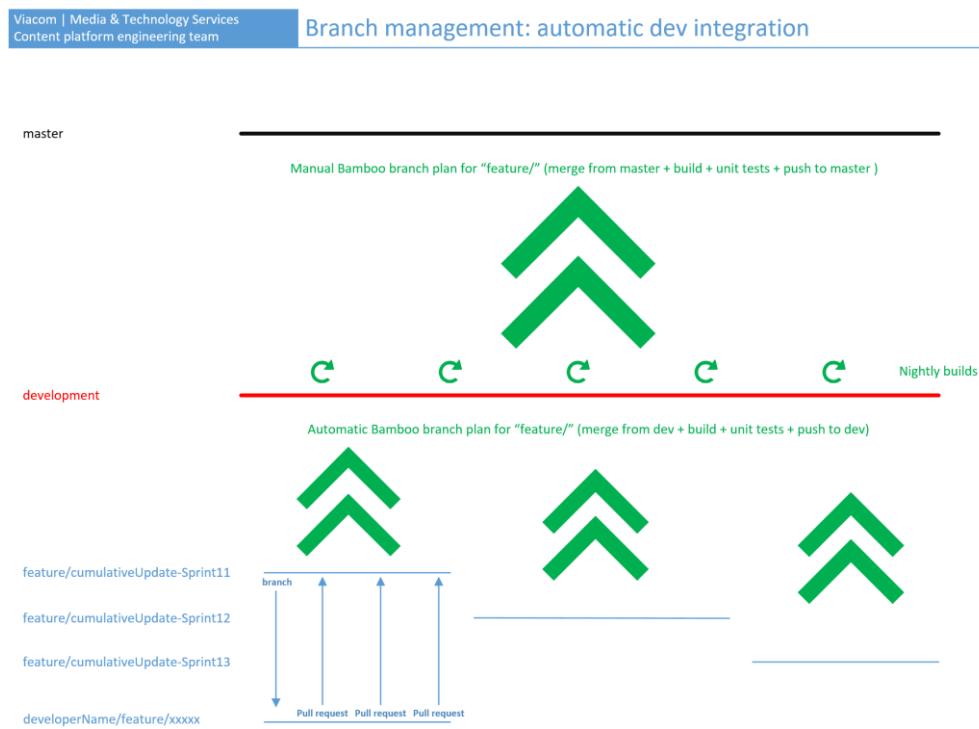
- He creates a “pull request”. The pull request is a way to say “Hey QA team, could you please review and integrate my code into the “feature/” branch?”
- The QA team or tech lead review your code, if good it is approved and integrated into the development branch. Otherwise it is rejected and you will 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

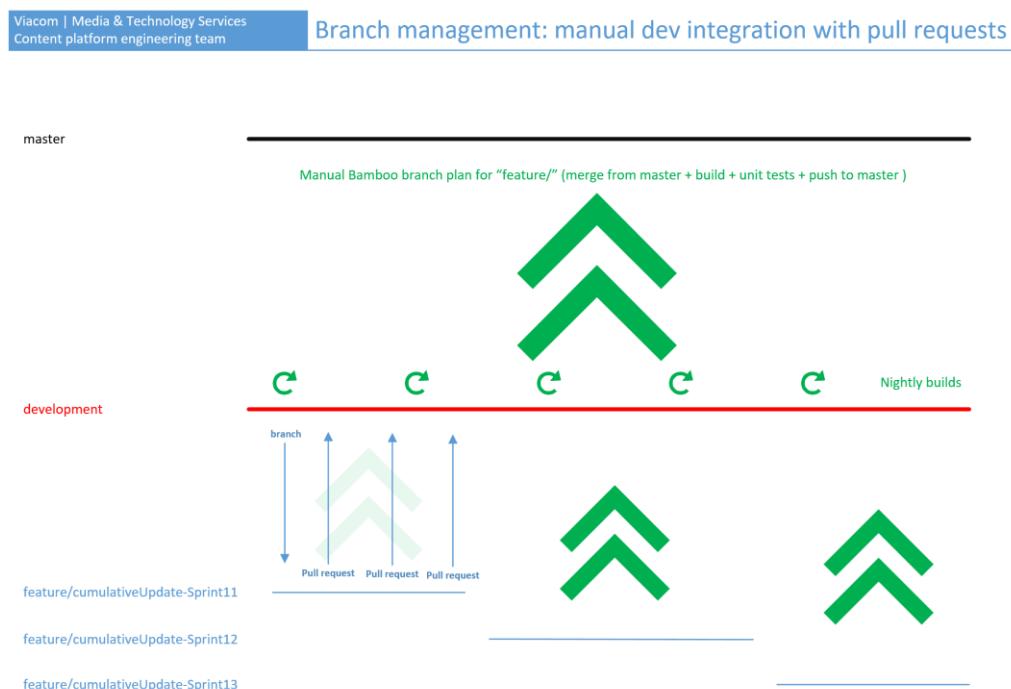
can be accomplish by implementing “pull requests” integration strategy in your development workflow. This is enabled from the main code repository in bitbucket.

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, enum, 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.

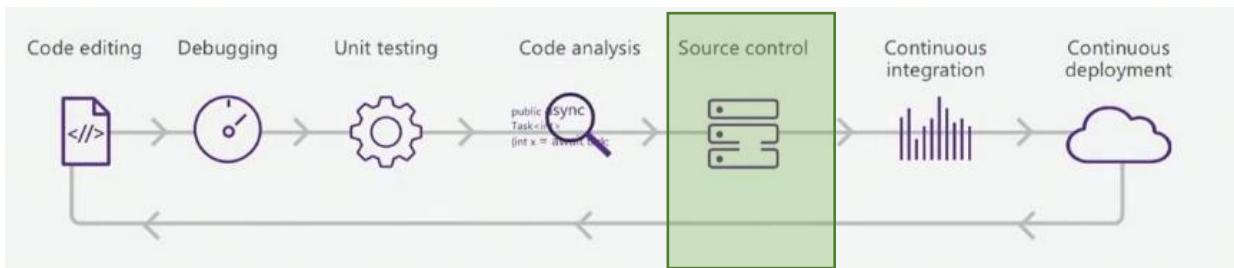
As a minimum base-line we define the following attributes that defines what is “good” code:

- 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](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.*

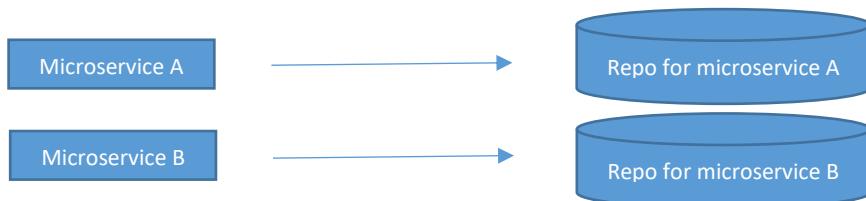
## Source code management

### Introduction



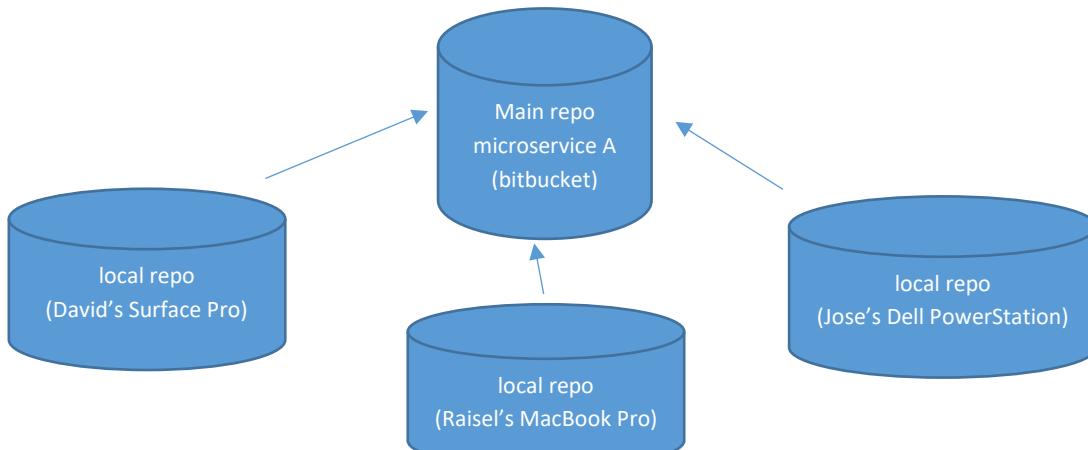
Source code management is the key part upon which the continuous delivery workflow is built. In this document, all examples are described in “git”, but it could be adapted in order to use other SCMs tools like mercurial (hg).

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.mtv.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
  - python
  - 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.mtv.com/projects/VCP/repos/vcp-netcore-webappdem01>

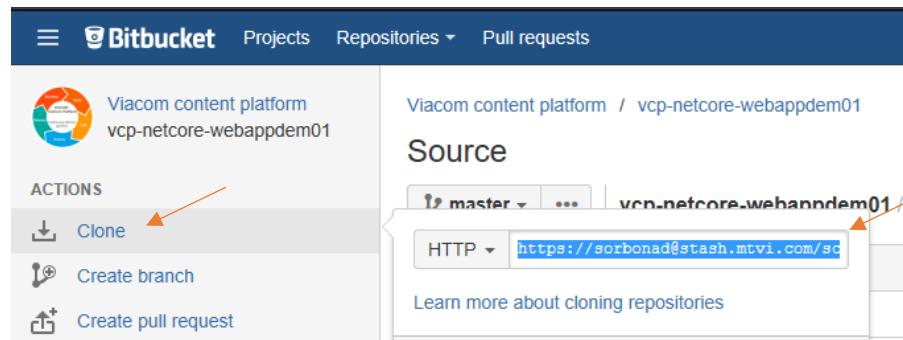
## Code repository setup in development unit

There 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:\vimm-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:\vimm-america_code-repository\Viacom.ContentPlatform  
git clone https://sorbonad@stash.mtv.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:\viamerica_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.

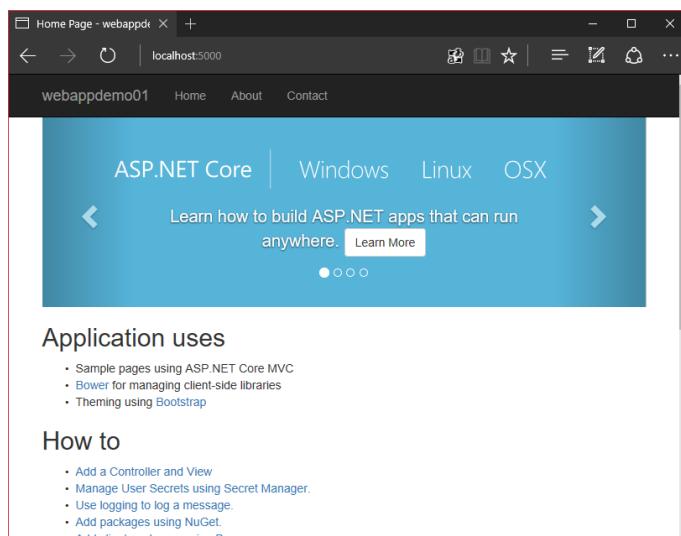
For changing branches, you just have to run this command:

```
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
```

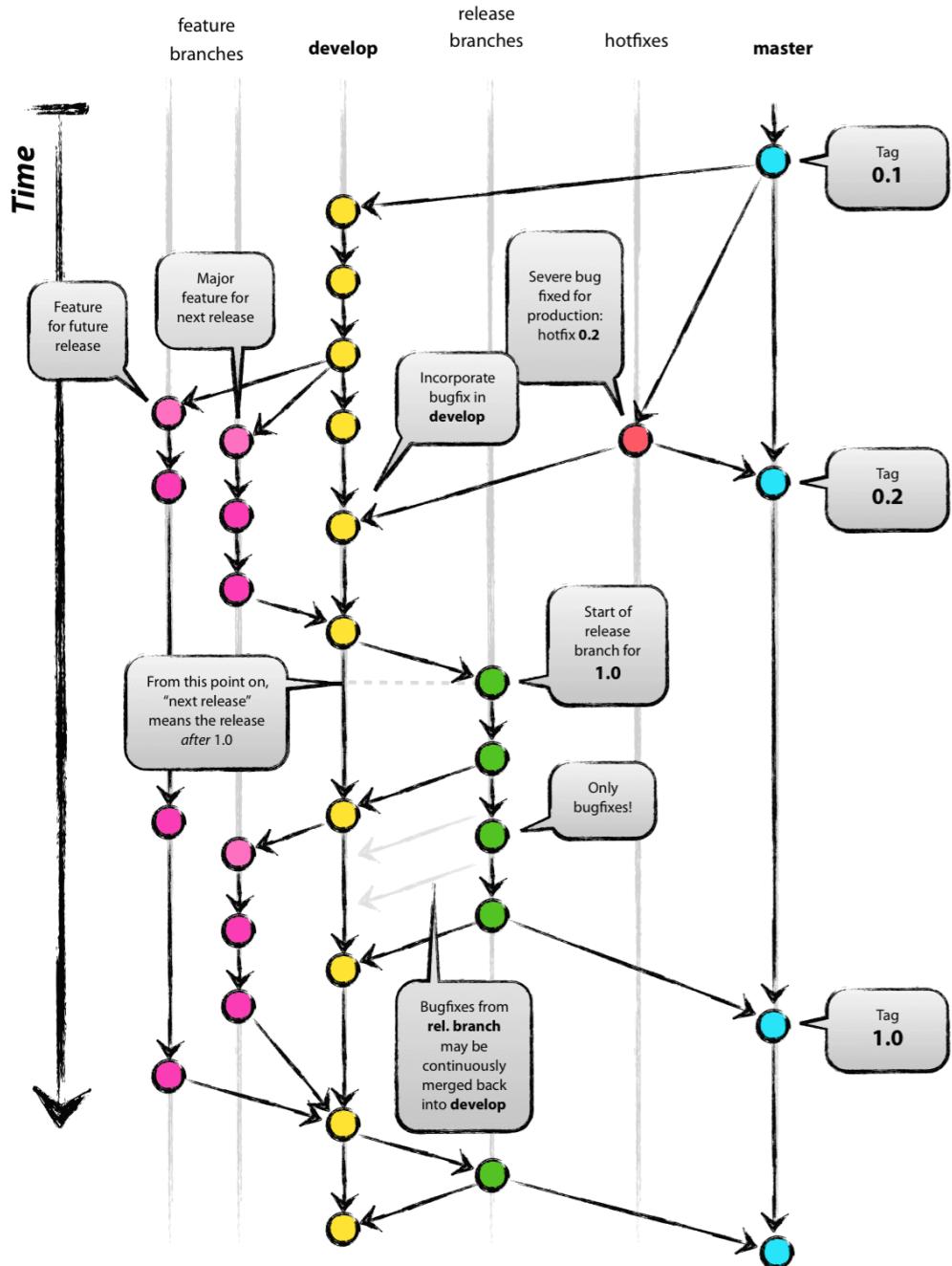
5. Done! Now your changeset will be hosted in bitbucket

6. Code something else, go to point 1.

## Branch management

THIS SECTION 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/>



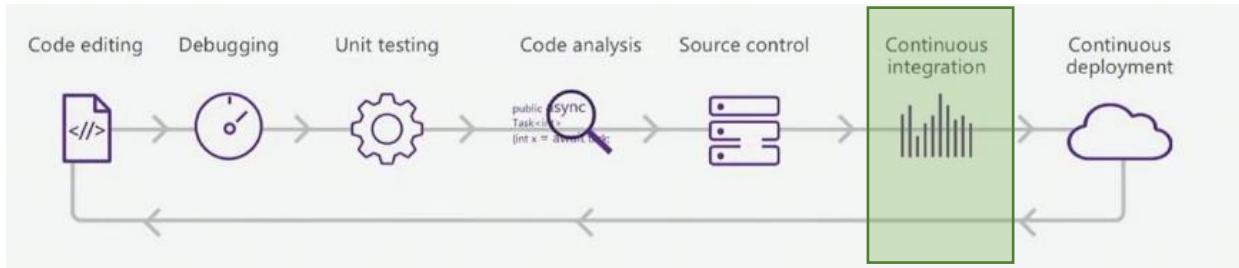
Currently the Viacom CD workflow supports four branches:

| Branch                            | Description   |
|-----------------------------------|---|
| master                            | <ul style="list-style-type: none"> <li>• Host the code that is in production.</li> <li>• You never develop this branch.</li> <li>• There is only 1.</li> <li>• Name: master</li> </ul>  |
| development                       | <ul style="list-style-type: none"> <li>• Integration branch where all changes are merged before going to master.</li> <li>• You never develop this branch.</li> <li>• There is only 1.</li> <li>• Name: development.</li> </ul>   |
| feature branches                  | <ul style="list-style-type: none"> <li>• Branch where you code something.</li> <li>• There could be several feature branches at the same time.</li> <li>• Once a commit from a feature branches is pushed, it is merge to the development branch</li> <li>• Naming convention: mandatory naming prefix: "feature/"</li> </ul>                                   |
| Experimental or personal branches | <ul style="list-style-type: none"> <li>• Where you can do experiments, play with the code, break things</li> <li>• There could be several experimental-personal branches.</li> <li>• Naming convention: mandatory naming prefix: "userName/" or "experimental/"</li> <li>• This branch can be pushed to main central repository for backup purposes.</li> </ul> |



## 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”

A screenshot of the 'Create a new plan' configuration page. The title is 'Create a new plan' and the sub-section is 'Configure plan'. A text block explains that the build plan defines everything about the build process. On the left, there's a sidebar with 'Project and build' and 'Plan name \*'. The 'Project and build' section has a dropdown menu with 'New Project' (highlighted with an orange arrow) and 'Existing Projects'. Under 'Existing Projects', 'Broadcast Systems Development' is selected. Other options include CAPI, COMPASS, and Crabapple.

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”

Create a new plan

[Configure plan](#) [Configure tasks](#)

### Configure plan

How to create a build plan

Your build plan defines everything about your build process. Each plan has a Default job when it is created. More advanced configuration options (including those for plugins), and the ability to add more jobs will be available to you after creating this plan.

**Project and build plan name**

Project  The project the new plan will be created in.

Project name\*  Eg. AT (for a project named Atlassian)

Project key\*  Eg. WEB (for a plan named Website)

Plan name\*

Plan key\*  Eg. WEB (for a plan named Website)

Plan description

**Link repository to new build plan**

Repository host\*  Previously linked repository  Link new repository

Bitbucket Server / Stash

Display name\*

**Bitbucket Server / Stash details**

Server

Repository

A new public key for the selected repository will be stored as a repository access key.

Branch

Repository access  Only you are allowed to reuse the configuration of this repository  Allow all users to reuse the configuration of this repository

[Configure plan](#) [Cancel](#)

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”

31 agents have the [capabilities](#) to run this job

**Source Code Checkout**

Checkout Default Repository

- Docker  
Build sdk-image to compile source code (Release)
- Script  
Create working subfolders if needed
- Script  
Run unit tests in docker sdk-container
- MSTest Parser  
Parse unit test results
- Script  
Remove sdk-container (safe guard)
- Command  
Create sdk-container layer to extract compiled binaries from sdk-image
- Command  
Copy compiled binaries from sdk-container to PublishOutput path
- Docker  
Build runtime optimized release-image
- Command  
Save release-image to build artifacts folder
- Command  
Remove old container (Optional)

**Source Code Checkout configuration**

How to use the Source Code Checkout task

Task description

Checkout Default Repository

Disable this task

You can check out one or more repositories with this Task. You can choose to check out the Plan’s *Default Repository* or specify a *Specific Repository*. You can add additional repositories to this Plan via the [Plan configuration](#).

Repository\*

vcp-netcore-webappdem01

Default always points to Plans default repository.

Checkout Directory

(Optional) Specify an alternative sub-directory to which the code will be checked out.

Force Clean Build

Removes the source directory and checks it out again prior to each build. This may significantly increase build times.

[+ Add repository](#)

**Save**   **Cancel**

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

**Source Code Checkout**

Checkout Default Repository

---

**Docker**

Build sdk-image to compile source code (Release)

---

**Script**

Create working subfolders if needed

---

**Script**

Run unit tests in docker sdk-container

---

**MSTest Parser**

Parse unit test results

---

**Script**

Remove sdk-container (safe guard)

---

**Command**

Create sdk-container layer to extract compiled binaries from sdk-image

---

**Command**

Copy compiled binaries from sdk-container to PublishOutput path

---

**Docker**

Build runtime optimized release-image

---

**Command**

Save release-image to build artifacts folder

---

**Command**

Remove sdk-container (Clean up)

---

**Command**

Remove sdk-image (Clean up)

---

**Command**

Remove release-image (Clean up)

---

**Final tasks** Are always executed even if a previous task fails

Drag tasks here to make them final

---

Add task

**Docker configuration**

Task description

Build sdk-image to compile source code (Release)

Disable this task

Command

Build a Docker image

The Docker command to execute

Repository\*

vcp-netcore-webappdem01:sdkimage

Repository name (and optionally a tag) to be applied to the resulting image (e.g. 'registry.address:port/namespace/repository:tag')

Dockerfile

Use an existing Dockerfile located in the task's working directory

Specify the Dockerfile contents

```
# Viacom content platform engineering team
# Created by Eng. David Sorbona
# v1.0 - 2017

FROM microsoft/aspnetcore-build
WORKDIR /app

ENV LTTNG_UST_REGISTER_TIMEOUT=0
```

Do not use cache when building the image

Save the image as a file

**Advanced options**

Environment variables

Extra environment variables. e.g. JAVA\_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space.

Working sub directory

Specify an alternative sub-directory as working directory for the task.

**Save** **Cancel**

### 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"

- Source Code Checkout  
Checkout Default Repository
- Docker  
Build sdk-image to compile source code (Release)
- Script  
Create working subfolders if needed
- Script  
Run unit tests in docker sdk-container
- MSTest Parser  
Parse unit test results
- Script  
Remove sdk-container (safe guard)
- Command  
Create sdk-container layer to extract compiled binaries from sdk-image
- Command  
Copy compiled binaries from sdk-container to PublishOutput path
- Docker  
Build runtime optimized release-image
- Command  
Save release-image to build artifacts folder
- Command  
Remove sdk-container (Clean up)
- Command  
Remove sdk-image (Clean up)
- Command  
Remove release-image (Clean up)
- Final tasks** Are always executed even if a previous task fails

Drag tasks here to make them final

Add task

### Script configuration

Task description

Create working subfolders if needed

Disable this task

Interpreter

/bin/sh or cmd.exe

Run your script with /bin/sh or cmd.exe

Script location

Inline

Script body\*

```

1 #!/bin/sh
2 mkdir -p ./build-artifacts
3 mkdir -p ./publish-binaries-output
4

```

Argument

Environment variables

Working sub directory

**Save** **Cancel**

## 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"

**Source Code Checkout**

Checkout Default Repository

---

**Docker**

Build sdk-image to compile source code (Release)

---

**Script**

Create working subfolders if needed

---

**Script**

Run unit tests in docker sdk-container and save result in xml

**MSTest Parser**

Parse unit test results

---

**Script**

Remove sdk-container (safe guard)

---

**Command**

Create sdk-container layer to extract compiled binaries from sdk-image

---

**Command**

Copy compiled binaries from sdk-container to PublishOutput path

---

**Docker**

Build runtime optimized release-image

---

**Command**

Save release-image to build artifacts folder

---

**Command**

Remove sdk-container (Clean up)

---

**Command**

Remove sdk-image (Clean up)

---

**Command**

Remove release-image (Clean up)

---

**Final tasks** Are always executed even if a previous task fails

Drag tasks here to make them final

---

Add task

### Script configuration

Task description

Run unit tests in docker sdk-container and save result in xml

Disable this task

Interpreter

Shell

An interpreter is chosen based on the shebang line of your script.

Script location

Inline

Script body\*

```
1 #!/bin/sh
2 docker run --volume ${bamboo_build_working_directory}:/data --workdir
3 exit 0
```

Argument

Environment variables

Working sub directory

**Save** **Cancel**

Script body:

```
#!/bin/sh
docker run --volume ${bamboo_build_working_directory}:/data --workdir /app --
rm vcp-netcore-webappdem01:sdkiimage 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"

|   |   |  |
|---|---|--|
| <div style="border: 1px solid #ccc; padding: 5px;"> <p> Source Code Checkout</p> <p> Docker</p> <p> Script</p> <p> Script</p> <p><b>MSTest Parser</b></p> <p> Parse unit test results</p> <p> Script</p> <p> Command</p> <p> Command</p> <p> Docker</p> <p> Command</p> <p> Command</p> <p> Command</p> <p> Command</p> <p> Command</p> <p> Final tasks</p> <p style="text-align: center;"><i>Drag tasks here to make them final</i></p> </div> | <p><b>MSTest Parser configuration</b></p> <p>Task description</p> <p><input type="text" value="Parse unit test results"/></p> <p><input type="checkbox"/> Disable this task</p> <p>MSTest Test Results File/Directory*</p> <p><input type="text" value="unit-tests-results.xml"/></p> <p>The test files must be in MSTest format.</p> <p><b>Advanced options</b></p> <p><input type="checkbox"/> Pick up test results that were created outside of this build<br/>Files created before the current build was started will be analyzed as valid tests results</p> <p><b>Save</b> <b>Cancel</b></p> | <p>How to use the MSTest Parser task</p> |
|---|---|--|

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)"

Source Code Checkout

Checkout Default Repository

Docker

Build sdk-image to compile source code (Release)

Script

Create working subfolders if needed

Script

Run unit tests in docker sdk-container and save result in xml

MSTest Parser

Parse unit test results

**Script**

Remove sdk-container (safe guard)

Command

Create sdk-container layer to extract compiled binaries from sdk-image

Command

Copy compiled binaries from sdk-container to PublishOutput path

Docker

Build runtime optimized release-image

Command

Save release-image to build artifacts folder

Command

Remove sdk-container (Clean up)

Command

Remove sdk-image (Clean up)

Command

Remove release-image (Clean up)

**Final tasks** Are always executed even if a previous task fails

Drag tasks here to make them final

Add task

### Script configuration

Task description

Disable this task

Interpreter

Run your script with /bin/sh or cmd.exe

Script location

Script body\*

```
1 #!/bin/sh
2 docker rm $(docker ps -aq --filter name=sdkcontainer)
3 exit 0
```

Argument

Environment variables

Working sub directory

**Save** **Cancel**

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"

|  |   |   |
|--|---|---|
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">Source Code Checkout</span><br/> Checkout Default Repository </div>                        | <span style="color: #0070C0;">Command configuration</span><br>Task description<br><div style="border: 1px solid #ccc; padding: 2px; margin-top: 5px;"> Create sdk-container layer to extract compiled binaries from sdk-image </div>                          | <a href="#" style="color: #0070C0;">How to use the Command task</a>   |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">Docker</span><br/> Build sdk-image to compile source code (Release) </div>                 | <input type="checkbox"/> Disable this task  |   |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">Script</span><br/> Create working subfolders if needed </div>                              | Executable<br><div style="border: 1px solid #ccc; padding: 2px; width: 150px; margin-top: 5px;"> docker </div>  |   |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">Script</span><br/> Run unit tests in docker sdk-container and save result in xml </div>    | Argument<br><div style="border: 1px solid #ccc; padding: 2px; width: 150px; margin-top: 5px;"> create --name sdkcontainer vcp-netcore-webappdem01:sdkimage --rm </div>  | <div style="border: 1px solid #ccc; padding: 2px; margin-top: 5px;"> Argument you want to pass to the command. Arguments with spaces in them must be quoted. </div> |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">MSTest Parser</span><br/> Parse unit test results </div>                                   |   |   |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">Script</span><br/> Remove sdk-container (safe guard) </div>                                |   |   |
| <span style="color: white;">Command</span><br>Create sdk-container layer to extract compiled binaries from sdk-image   | Environment variables<br><div style="border: 1px solid #ccc; padding: 2px; margin-top: 5px;"></div>   | Extra environment variables. e.g. JAVA_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space.  |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">Command</span><br/> Copy compiled binaries from sdk-container to PublishOutput path </div> | Working sub directory<br><div style="border: 1px solid #ccc; padding: 2px; margin-top: 5px;"></div>   | Working sub directory<br><div style="border: 1px solid #ccc; padding: 2px; margin-top: 5px;"></div>   |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">Docker</span><br/> Build runtime optimized release-image </div>                            | <div style="text-align: right; margin-top: 5px;"> <span style="border: 1px solid #0070C0; padding: 2px 10px; border-radius: 5px; background-color: #0070C0; color: white; cursor: pointer;">Save</span> <span style="margin-left: 10px;">Cancel</span> </div> |   |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">Command</span><br/> Save release-image to build artifacts folder </div>                    |   |   |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">Command</span><br/> Remove sdk-container (Clean up) </div>                                 |   |   |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">Command</span><br/> Remove sdk-image (Clean up) </div>                                     |   |   |
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">Command</span><br/> Remove release-image (Clean up) </div>                                 |   |   |
| <b>Final tasks</b> Are always executed even if a previous task fails   |   |   |
| <i>Drag tasks here to make them final</i>  |   |   |
| <a href="#" style="border: 1px solid #ccc; padding: 2px 10px; border-radius: 5px; text-decoration: none; color: inherit;">Add task</a>   |   |   |

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"

| <pre> Source Code Checkout   Checkout Default Repository  Docker   Build sdk-image to compile source code (Release)  Script   Create working subfolders if needed  Script   Run unit tests in docker sdk-container and save result     in xml  MSTest Parser   Parse unit test results  Script   Remove sdk-container (safe guard)  Command   Create sdk-container layer to extract compiled     binaries from sdk-image  <b>Command</b>   Copy compiled binaries from sdk-container to     PublishOutput path  Docker   Build runtime optimized release-image  Command   Save release-image to build artifacts folder  Command   Remove sdk-container (Clean up)  Command   Remove sdk-image (Clean up)  Command   Remove release-image (Clean up) </pre> | <p><b>Command configuration</b></p> <p>Task description</p> <p><input type="text" value="Copy compiled binaries from sdk-container to PublishOutput path"/> <span style="float: right;">×</span></p> <p><input type="checkbox"/> Disable this task</p> <p>Executable</p> <p><input type="text" value="docker"/> <span style="float: right;">▼</span></p> <p>Argument</p> <p><input type="text" value="cp sdkcontainer:/out ./publish-binaries-output"/></p> <p>Argument you want to pass to the command. Arguments with spaces in them must be quoted.</p> <p>Environment variables</p> <p><input type="text"/></p> <p>Extra environment variables. e.g. JAVA_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space.</p> <p>Working sub directory</p> <p><input type="text"/></p> <p>Specify an alternative sub-directory as working directory for the task.</p> <p style="text-align: center;"><span style="background-color: #0070C0; color: white; padding: 2px 10px; border-radius: 5px;">Save</span> <span style="border: 1px solid #ccc; padding: 2px 10px; border-radius: 5px;">Cancel</span></p> | <p>How to use the Command task</p> |
|--|---|------------------------------------|
| <p><b>Final tasks</b> Are always executed even if a previous task fails</p> <p>Drag tasks here to make them final</p> <hr/> <p>Add task</p>  |   |                                    |

Argument:

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

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



nickelodeon



telefe

## 9. "Docker / Build runtime optimized release-image"

**Docker configuration**

Task description

Disable this task

Command

The Docker command to execute

Repository\*

Repository name (and optionally a tag) to be applied to the resulting image (e.g. 'registry.address:port/namespace/repository:tag')

Dockerfile

Use an existing Dockerfile located in the task's working directory  
 Specify the Dockerfile contents

```
# Viacom content platform engineering team
# Created by Eng. David Sorbona
# v1.0 - 2017

FROM microsoft/aspnetcore
WORKDIR /app

COPY ./publish-binaries-output .
```

Do not use cache when building the image  
 Save the image as a file

**Advanced options**

Environment variables

Extra environment variables. e.g. JAVA\_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space.

Working sub directory

Specify an alternative sub-directory as working directory for the task.

**Save** **Cancel**

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"

|   |   |                                    |
|---|---|------------------------------------|
| <p><b>Source Code Checkout</b><br/>Checkout Default Repository</p> <p><b>Docker</b><br/>Build sdk-image to compile source code (Release)</p> <p><b>Script</b><br/>Create working subfolders if needed</p> <p><b>Script</b><br/>Run unit tests in docker sdk-container and save result in xml</p> <p><b>MSTest Parser</b><br/>Parse unit test results</p> <p><b>Script</b><br/>Remove sdk-container (safe guard)</p> <p><b>Command</b><br/>Create sdk-container layer to extract compiled binaries from sdk-image</p> <p><b>Command</b><br/>Copy compiled binaries from sdk-container to PublishOutput path</p> <p><b>Docker</b><br/>Build runtime optimized release-image</p> <p><b>Command</b><br/><b>Save release-image to build artifacts folder</b></p> <p><b>Command</b><br/>Remove sdk-container (Clean up)</p> <p><b>Command</b><br/>Remove sdk-image (Clean up)</p> <p><b>Command</b><br/>Remove release-image (Clean up)</p> <p><b>Final tasks</b> Are always executed even if a previous task fails</p> | <p><b>Command configuration</b></p> <p>Task description<br/><b>Save release-image to build artifacts folder</b></p> <p><input type="checkbox"/> Disable this task</p> <p>Executable<br/>docker</p> <p>Argument<br/>save --output ./build-artifacts/vcp-netcore-webappdem01-release-image-artifact.tar</p> <p>Argument you want to pass to the command. Arguments with spaces in them must be quoted.</p> <p>Environment variables</p> <p>Extra environment variables. e.g. JAVA_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space.</p> <p>Working sub directory</p> <p>Specify an alternative sub-directory as working directory for the task.</p> <p><b>Save</b>   <a href="#">Cancel</a></p> | <p>How to use the Command task</p> |
|---|---|------------------------------------|

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)"

The screenshot shows a build configuration interface with a sidebar of available tasks and a main panel for configuring a selected task. The sidebar includes: Source Code Checkout, Docker, Script, MSTest Parser, Script, Command, Command, Docker, Command, Final tasks. The main panel for the selected 'Command' task shows:

- Task description:** Remove sdk-container (Clean up)
- Executable:** docker
- Argument:** rm -f sdkcontainer
- Environment variables:** (empty)
- Working sub directory:** (empty)

Buttons at the bottom right include Save and Cancel.

Argument: rm -f sdkcontainer

## 12. "Command / Remove sdk-image (Clean up)"

The screenshot shows a build configuration interface with a sidebar of available tasks and a main panel for configuring a selected task. The sidebar includes: Source Code Checkout, Docker, Script, MSTest Parser, Script, Command, Command, Docker, Command, Final tasks. The main panel for the selected 'Command' task shows:

- Task description:** Remove sdk-image (Clean up)
- Executable:** docker
- Argument:** rmi -f vcp-netcore-webappdem01:sdkimage
- Environment variables:** (empty)
- Working sub directory:** (empty)

Buttons at the bottom right include Save and Cancel.

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

### 13. "Command / Remove release-image (Clean up)"

|  |   |
|--|---|
| Source Code Checkout   | X |
| Checkout Default Repository  |   |
| Docker   | X |
| Build sdk-image to compile source code (Release)                       |   |
| Script   | X |
| Create working subfolders if needed                                    |   |
| Script   | X |
| Run unit tests in docker sdk-container and save result in xml          |   |
| MSTest Parser  | X |
| Parse unit test results  |   |
| Script   | X |
| Remove sdk-container (safe guard)                                      |   |
| Command  | X |
| Create sdk-container layer to extract compiled binaries from sdk-image |   |
| Command  | X |
| Copy compiled binaries from sdk-container to PublishOutput path        |   |
| Docker   | X |
| Build runtime optimized release-image                                  |   |
| Command  | X |
| Save release-image to build artifacts folder                           |   |
| <b>Final tasks</b> Are always executed even if a previous task fails   |   |
| Command  | X |
| Remove sdk-container (Clean up)  |   |
| Command  | X |
| Remove sdk-image (Clean up)  |   |
| <b>Command</b>   | X |
| <b>Remove release-image (Clean up)</b>                                 |   |

[Add task](#)

**Command configuration**

Task description  
Remove release-image (Clean up)

Disable this task

Executable  
docker

Argument  
rmi -f vcp-netcore-webappdem01:release

Argument you want to pass to the command. Arguments with spaces in them must be quoted.

Environment variables

Extra environment variables. e.g. JAVA\_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space.

Working sub directory

Specify an alternative sub-directory as working directory for the task.

[Save](#) [Cancel](#)

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

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

|  |   |
|--|---|
| Command  | X |
| Save release-image to build artifacts folder                         |   |
| <b>Final tasks</b> Are always executed even if a previous task fails |   |
| Command  | X |
| Remove sdk-container (Clean up)                                      |   |
| Command  | X |
| Remove sdk-image (Clean up)  |   |
| Command  | X |
| Remove release-image (Clean up)                                      |   |

[Add task](#)

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:

| 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:

The screenshot shows the Bamboo web interface for managing a CI/CD plan. The URL is [Build projects / Viacom content platform / vcp-netcore-webappdem01 Configuration - vcp-netcore-webappdem01](#). The left sidebar shows project navigation. The main area is titled "Configuration - vcp-netcore-webappdem01". It displays the "Plan Configuration" section, which is part of the "Viacom content platform continuous delivery pipeline model". The "Plan Configuration" tab is highlighted with a blue bar. Below it, there are tabs for "Plan details", "Stages", "Repositories", "Triggers", "Branches", "Dependencies", "Permissions", "Notifications", "Variables", and "Miscellaneous". The "Plan details" tab is active. The "Plan details" section contains fields for "Project name" (Viacom content platform), "Plan name" (vcp-netcore-webappdem01), and "Plan description" (CI/CD demo plan part of the Viacom content platform continuous delivery pipeline model). A checkbox labeled "Plan enabled" is checked, with an orange arrow pointing to it. At the bottom are "Save" and "Cancel" buttons.

2. Rename the stage as “Build”:

The screenshot shows the 'Plan Configuration' screen for a plan named 'Build'. On the left, there's a sidebar with sections for 'Stages & jobs' (containing 1 item), 'Build' (containing 1 job), and 'Branches' (containing 3 branches: 'development', 'development (merge into master)', and 'feature-NewValidation01'). The main area is titled 'Plan contents' and describes stages as steps in a build process. A specific stage named 'Build' is highlighted with a blue border. To the right of the stage, a context menu is open, with the 'Configure stage' option highlighted in blue and an orange arrow pointing to it.

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:

The screenshot shows the 'Plan Configuration' screen for the same 'Build' plan. The 'Triggers' tab is selected. It shows a single trigger named 'Bitbucket Server repository triggered'. This trigger is of type 'Scheduled' with a 'Daily build (6:00 AM)' configured. To the right of the trigger, there's a 'Trigger configuration' panel with fields for 'Trigger description' (empty), 'Disable this trigger' (unchecked), and 'Trigger conditions' (unchecked). At the bottom of the panel are 'Save trigger' and 'Cancel' buttons. Orange arrows point from the text labels 'Bitbucket Server repository triggered' and 'Save trigger' to their respective elements on the screen.

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

**Plan Configuration**

Stages & jobs 1

**Build**

- Build + test + containerization + publish artifact

**Branches** 3

- development
- development (merge into master)
- feature-NewValidation01

**Triggers**

If you want Bamboo to start this plan automatically, you will need to set triggers to specify how and when the build will at any time, use the "run" menu or trigger a release from JIRA.

Bitbucket Server repository triggered

Scheduled

Daily build (6:00 AM)

Add trigger

Trigger configuration

Trigger description

Daily build (6:00 AM)

Disable this trigger

Schedule\*

Daily at 6:00 am

Trigger conditions

Only run Build if other Plans are currently passing

**Save trigger** Cancel

#### 4. Setup the “Permissions”:

**Plan Configuration**

Stages & jobs 1

**Build**

- Build + test + containerization + publish artifact

**Branches** 3

- development
- development (merge into master)
- feature-NewValidation01

**Permissions**

✓ Plan configuration saved successfully.

You can edit your plan permissions here. Permissions can be granted to specific users or groups.

Add user Add group

|                              | View                                | Edit                                | Build                               | Clone                               | Admin                               |
|------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| <b>Users</b>                 |                                     |                                     |                                     |                                     |                                     |
| David Sorbona                | <input checked="" type="checkbox"/> |
| <b>Groups</b>                |                                     |                                     |                                     |                                     |                                     |
| content_platform_engineering | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input type="checkbox"/>            |
| <b>Other</b>                 |                                     |                                     |                                     |                                     |                                     |
| Logged in users              | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/>            |
| Anonymous users              | <input checked="" type="checkbox"/> |                                     |                                     |                                     |                                     |
| <b>Save</b> Cancel           |                                     |                                     |                                     |                                     |                                     |

## 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”

Build projects / Viacom content platform / vcp-netcore-webappdem01  
**Configuration - vcp-netcore-webappdem01**

CI/CD demo plan part of the Viacom content platform continuous delivery pipeline model

Plan Configuration

Stages & jobs 1

**Build**

Build + test + containerization + publish artifact

Branches 3

development

development (merge into master)

feature-NewValidation01

Job details Tasks Requirements Artifacts Miscellaneous

**Edit job details**

Job name \* Build + test + containerization + publis

Job description

Job enabled  
Disabling a job will prevent it from being run as part of the plan.

Save Cancel

2. Create an “artifact definition”

Plan Configuration

Stages & jobs 1

**Build**

Build + test + containerization + publish artifact

Branches 3

development

Artifact definitions

Create artifact definitions for artifacts you want to keep or share with others builds and deployments (e.g. JAR files, reports etc).

| Name                   | Location        | Copy pattern | Operations              |
|------------------------|-----------------|--------------|-------------------------|
| release-image-artifact | build-artifacts | *            | Unshare   Edit   Delete |

Create definition

Then complete the form with these values:

Name **\***  If the artifact is shared, the name must be unique within the plan

Location  Specify the directory (relative path) to find your artifact. e.g. target

Copy pattern **\***  Specify the name (or [Ant file copy pattern](#)) of the artifact(s) you want to keep. e.g. \*\*/\*jar

Shared  
Make the artifact available to be used in other builds and deployments.

## 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 .\vimm-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 or 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:\vimm-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:\vimm-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>



nickelodeon



telefe

## 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 than 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 (merge 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.

### **WARNING**

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:

**Automatic branch management**

Plan branches can be created and deleted automatically based on branch creation and deletion in the primary source repository.

|                   |  |
|-------------------|--|
| New branches      | Create plan branches for matching new branches               |
| Match name        | <code>^(feature hotfix release)V.+</code>                    |
| Deleted branches  | Delete plan branches after a period of time<br>7 days        |
| Inactive branches | Delete plan branches after a period of inactivity<br>30 days |

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

### Merging

Automatic merging can test the merge between branches and push changes back to the repository on a successful build. This setting will be applied to all new plan branches.

Branch merging enabled

|                |                |
|----------------|----------------|
| Branch updater | Gatekeeper     |
| Checkout       | development    |
| Merge from     | Current branch |
| Build          | Merge result   |
| Push on        | Current branch |

### JIRA feature branches

Bamboo will automatically link a plan branch to a JIRA issue if the branch name contains a JIRA issue key. Bamboo can also create a remote link from the JIRA issue back to Bamboo so you can see the status of your branch in JIRA

Create remote links from JIRA issues to plan branches

**Notifications**

Notification preferences are applied to all new plan branches.

- Notify committers and people who have favourited the created branch

All committers and people who have favourited the branch will be notified for all build failures and the first successful build.

- Use the plan's notification settings

Use the same notification rules as configured for this plan

- Notifications should not be sent for the created branch

No notifications will be sent for the created branch.

**Triggers**

How new plan branches should be triggered.

Branch triggers

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”

Plan Configuration

- Stages & jobs 1
- Build
  - Build + test + containerization + publish

Audit log

Branches

Create plan branch

2. Completes the form as shown and save it

Branch details

Source repository Notifications Variables

**Branch details**

You can update the names and description of the current branch.

Display name \* development

How do you want to identify the new branch?

Branch description CI/CD demo plan part of the Viacom content platform continuous delivery pipeline

Choose a meaningful description for the new branch. For example, "Bamboo tasks feature branch".

Branch enabled

Clean up plan branch automatically

This plan branch will be removed after 7 days of being deleted, or after 30 days of inactivity.

**Trigger type**

Change trigger ?

Change the way that Bamboo triggers this plan branch

**Merging**

Enable automatic merging to integrate changes between the branch and the mainline, as well as (optionally) push the merge back to the repository upon a successful integrated build.

Branch merging enabled ?

Shallow clones will be disabled when performing branch integration.

|  |  |
|--|--|
| <input type="radio"/> Branch updater ?   | <input checked="" type="radio"/> Gatekeeper ?  |
| Checkout <input type="text" value="development"/>                                    | Checkout <input type="text" value="vcp-netcore-webappdem01"/>                                    |
| Merge from <input type="text" value="vcp-netcore-webappdem01"/>                      | Merge from <input type="text" value="development"/>  |
| Build Merge result   | Build Merge result   |
| Push on <input checked="" type="checkbox"/> <input type="text" value="development"/> | Push on <input checked="" type="checkbox"/> <input type="text" value="vcp-netcore-webappdem01"/> |

**Save** **Cancel**



Source repository

Source repository Bitbucket Server / Stash

Repository display name vcp-netcore-webappdem01

Branch development

- 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 your 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.

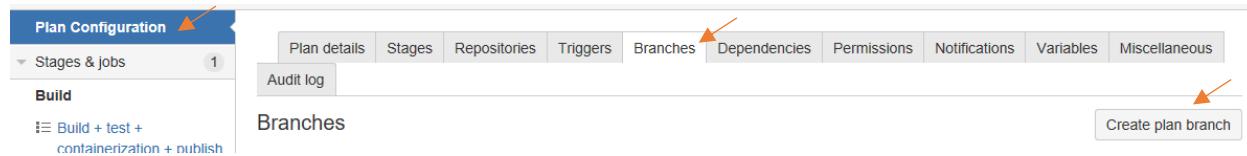
**delayed flight** code that is flying to production (master branch is updated) but for some reason you have paused the trip.

**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

The screenshot shows the 'Branch details' configuration form. It includes sections for 'Branch details', 'Trigger type', 'Trigger conditions', 'Merging', and two side-by-side 'Gatekeeper' configuration boxes.

- Branch details:** Fields include 'Display name' (set to 'development (merge into master)'), 'Branch description' (set to 'CI/CD demo plan part of the Viacom content platform continuous delivery pipeline'), 'Branch enabled' (checked), 'Clean up plan branch automatically' (checked), and 'Trigger type' (set to 'Manual').
- Trigger type:** A red arrow points to the dropdown menu.
- Trigger conditions:** A red arrow points to the checkbox 'Only run Build if other Plans are currently passing'.
- Merging:** A red arrow points to the checkbox 'Branch merging enabled'.
- Gatekeeper Configuration:** Two side-by-side boxes show different configurations:
  - Left Box:** 'Branch updater' selected. Fields: 'Checkout' (set to 'development (merge into master)'), 'Merge from' ('vcp-netcore-webappdem01'), 'Build' ('Merge result'), and 'Push on' (checkbox checked).
  - Right Box:** 'Gatekeeper' selected. Fields: 'Checkout' ('vcp-netcore-webappdem01'), 'Merge from' ('development (merge into master)'), 'Build' ('Merge result'), and 'Push on' (checkbox checked).

At the bottom are 'Save' and 'Cancel' buttons.

Branch details   Source repository   Notifications   Variables

Source repository Bitbucket Server / Stash

Repository display name vcp-netcore-webappdem01

Branch development

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 your 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

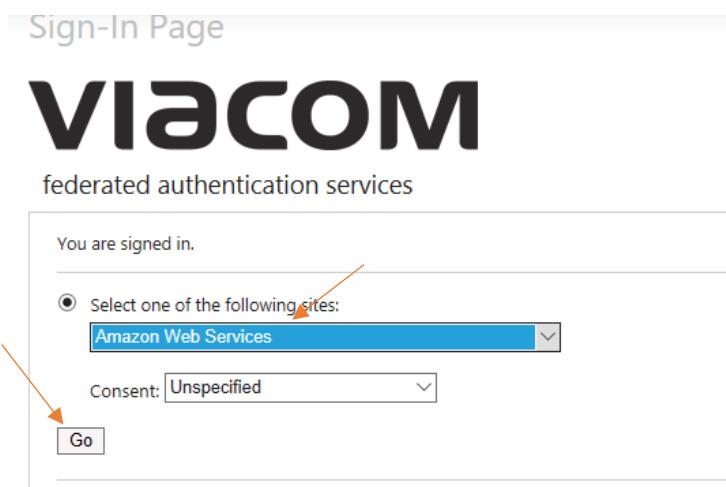
In order to do this, you need access to the Viacom AWS instance. The access is managed by the MCS team (Multiplatform cloud services)

### AWS ECR setup

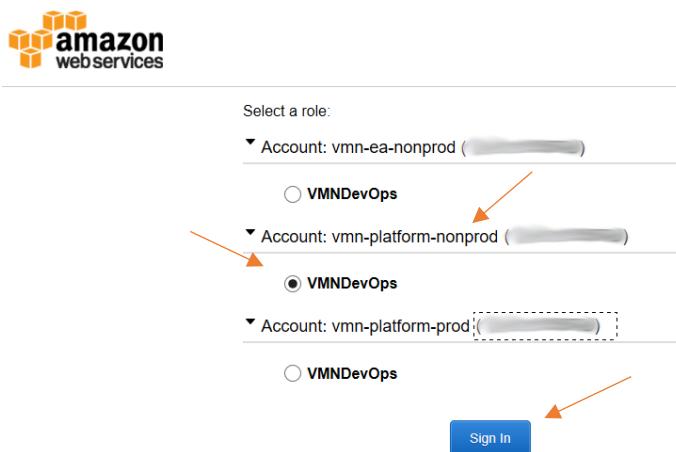
The follow steps show how to create the repository in the Viacom non-prod AWS environment:

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

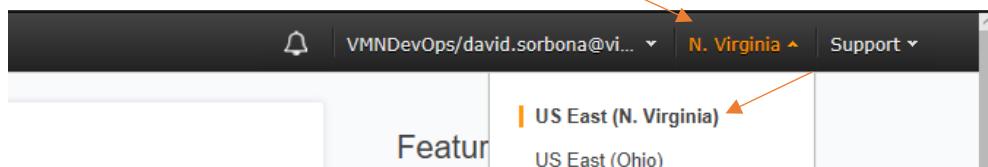
<https://fs.viacomcloud.com/adfs/ls/IdpInitiatedSignOn.aspx>



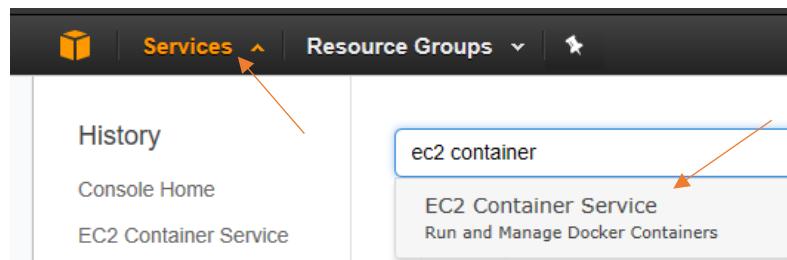
2. Select the account “vmn-platform-nonprod” and sign in.



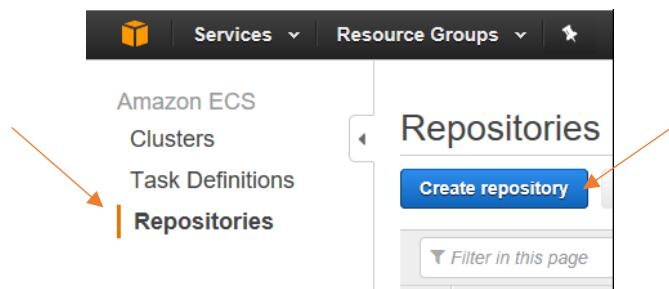
3. Go to region “North Virginia (us-east-1)”



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**

### Get started with EC2 Container Registry

**Step 1: Configure repository**

Step 2: Build, tag, and push Docker image

**Configure repository**

This wizard will guide you through the steps of creating a repository in EC2 Container Registry. [Learn more](#)

**Repository name\*** viacomcontentplatform/vcp-netcore-web

Namespaces are optional, and they can be included in the repository name with a slash (for example, namespace/repo)

**Repository URI** 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01

**Permissions**

As the owner, you have access to this repository by default. After completing this wizard, you can grant others permission to access this repository in the console.

\*Required Cancel **Next Step**

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.

### Get started with EC2 Container Registry

**Step 1: Configure repository**

**Step 2: Build, tag, and push Docker image**

Now that your repository exists, you can push a Docker image by following these steps.

**Successfully created repository**  
310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01

To install the AWS CLI and Docker and for more information on the steps below, visit the ECR [documentation page](#).

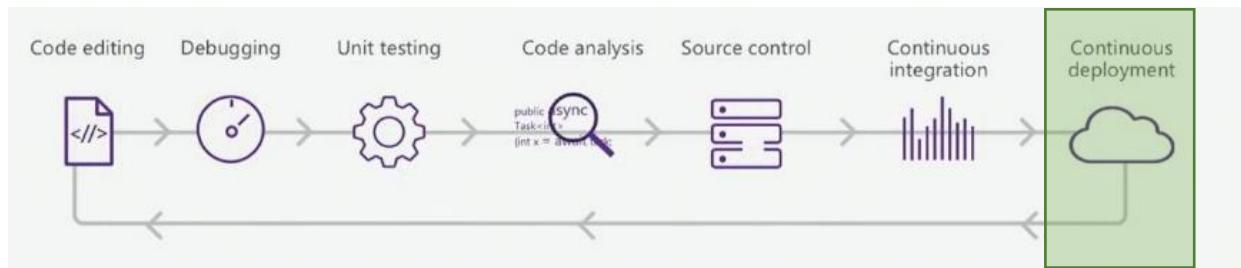
- 1) Retrieve the docker login command that you can use to authenticate your Docker client to your registry:  
`aws ecr get-login --region us-east-1`
- 2) Run the docker login command that was returned in the previous step.
- 3) Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions [here](#). You can skip this step if your image is already built:  
`docker build -t viacomcontentplatform/vcp-netcore-webappdem01 .`
- 4) After the build completes, tag your image so you can push the image to this repository:  
`docker tag viacomcontentplatform/vcp-netcore-webappdem01:latest 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:latest`
- 5) Run the following command to push this image to your newly created AWS repository:  
`docker push 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:latest`

\*Required Done



## Continuous deployment pipeline

### Introduction



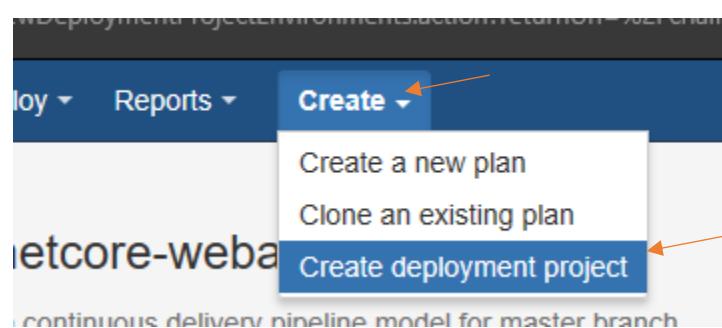
In the previous sections, we saw how to create an artifact (the “release-image” container) that will be deployed in some execution environment (AWS). This section will show how to publish that artifact to a container image repository by creating “deploy projects” in Bamboo.

### 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

Create deployment project How deployments work

A deployment project defines which build plan you get your artifacts from, and contains the environments you want to deploy to.

**Deployment project details**

|             |   |   |
|-------------|---|---|
| Name *      | Viacom content platform - vcp-netcore   | ← |
| Description | /viacom content platform continuous delivery pipeline model for master branch |   |

**Link to build plan** How deployment releases work

Shared artifacts of the selected plan will be bundled into releases. Releases will be deployed to the environments.

|   |   |   |
|---|---|---|
| Build plan *  | Viacom content platform > vcp-netcore-webappdem01 | ← |
| Start typing the plan name or use the down arrow to select a plan. The selected plan will be used as the source for artifacts that will be deployed as a release. |   |   |
| <input checked="" type="radio"/> Use the main plan branch ←<br>Currently  master<br><input type="radio"/> Use a custom plan branch                                |   |   |

[Create deployment project](#) [Cancel](#)

Naming convention:

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

In this example:

"Viacom content platform - vcp-netcore-webappdem01-master"

### 3. Set release versioning

Release versioning: Viacom content platform - vcp-netcore-webappdem01-master [How release versioning works](#)

Specify what version Bamboo should assign to automatically created releases. You can override this manually whenever you create a new release. Releases from branches will default to using the branch name suffixed with the build number of the build result.

**Next release**

Version \*

Generate preview

What version should Bamboo use for the next release? e.g. 1.0-m1 or 1.0-\${bamboo.buildNumber}

Add variable to version

**Automatically increment with each new release**

If you can't find variables in the list below that you added to the version, press 'Generate preview'. Bamboo will automatically add a suffix to non-unique versions.

Numbers  Last number in version

Variables No incrementable variables found in your version

Only plan variables and global variables are incremental.  
Password variables are excluded because they are not allowed in release version names.

**Preview**

Version of next release build-\${bamboo.buildNumber}.release-20

Version of subsequent release build-\${bamboo.buildNumber}.release-21

Why don't some variables show in the preview?  
Bamboo can only substitute variables in the preview if they do not depend on other data. If a variable depends on other workflows (e.g. \${bamboo.buildNumber}), it will be replaced at runtime but not in the preview.

**Save** **Cancel**

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.

Set up environment for Viacom content platform - vcp-netcore-webappdem01-master [How environments work](#)

Environments represent where releases are deployed to.

**Environment details**

Environment name \*

e.g. Staging, QA, or Production

Description

**Continue to task setup** **Create and back** **Cancel**

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 ECR. 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”

Clean working directory task configuration

Task description: Clean working directory

Disable this task

**Save** Cancel

2. “Artifact download / Download release contents”

Artifact download configuration

Task description: Download release contents

Disable this task

Artifact name\*: All artifacts

Destination path: /artifacts

Location that artifacts will be downloaded to, relative to the working directory

Add another artifact

**Save** Cancel

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

### 3. "Command / Load release-image from artifacts"

|   |  |
|---|--|
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[Clean working directory task]</span><br/> <span>Clean working directory</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[Artifact download]</span><br/> <span>Download release contents</span> </div> <div style="background-color: #0070C0; color: white; padding: 5px; margin-bottom: 5px;"> <span style="color: white;">[Command]</span><br/> <span>Load release-image from artifacts</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[Script]</span><br/> <span>Get AWS ECR Login info and export variables</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[Script]</span><br/> <span>Tag container image for AWS ECR</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[Command]</span><br/> <span>Publish container to AWS ECR</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[Command]</span><br/> <span>Remove release-image (Clean up)</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <b>Final tasks</b> Are always executed even if a previous task fails<br/><br/> <i>Drag tasks here to make them final</i> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px; text-align: center;"> <span>Add task</span> </div> | <p><b>Command configuration</b></p> <p>Task description</p> <div style="border: 1px solid #ccc; padding: 2px; width: 600px; margin-bottom: 5px;"> Load release-image from artifacts </div> <p><input type="checkbox"/> Disable this task</p> <p>Executable</p> <div style="border: 1px solid #ccc; padding: 2px; width: 200px; margin-bottom: 5px;"> <span style="color: orange;">docker</span> </div> <p>Argument</p> <div style="border: 1px solid #ccc; padding: 2px; width: 600px; margin-bottom: 5px;"> load --input /artifacts/vcp-netcore-webappdem01-release-image-artifact.tar </div> <p>Argument you want to pass to the command. Arguments with spaces in them must be quoted.</p> <p>Environment variables</p> <div style="border: 1px solid #ccc; padding: 2px; width: 600px; margin-bottom: 5px;"></div> <p>Extra environment variables. e.g. JAVA_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space.</p> <p>Working sub directory</p> <div style="border: 1px solid #ccc; padding: 2px; width: 600px; margin-bottom: 5px;"></div> <p>Specify an alternative sub-directory as working directory for the task.</p> <div style="text-align: right; margin-top: 10px;"> <span style="color: #0070C0; background-color: #0070C0; color: white; padding: 2px 10px; border-radius: 5px;">Save</span> <span>Cancel</span> </div> |
|---|--|

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.

The screenshot shows the Bamboo interface for configuring a script task. The task is titled "Get AWS ECR Login info and export variables". The configuration includes:

- Interpreter:** /bin/sh or cmd.exe
- Script location:** Inline
- Script body:** (contains the provided AWS CLI script)

#### Script body

```

AWS_STS_RESULTS=$(aws sts assume-role --role-arn
arn:aws:iam::${bamboo.awsAccountNumber}:role/${bamboo.awsECRole} --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 configuration**

Task description  
Tag container image for AWS ECR

Disable this task

Interpreter  
**Shell**

An interpreter is chosen based on the shebang line of your script.

Script location  
**Inline**

Script body\*

```
1 $(docker tag vcp-netcore-webappdem01:release 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:${bamboo.deploy.version})
2 $(docker tag vcp-netcore-webappdem01:release 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:latest)
```

How to use the Script task

Final tasks Are always executed even if a previous task fails

Drag tasks here to make them final

Add task

Argument

Environment variables

Working sub directory

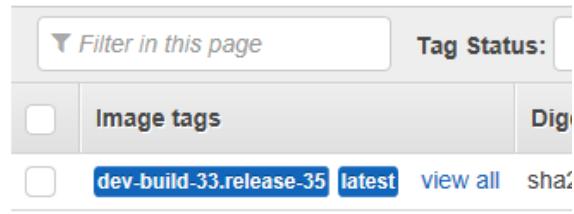
**Save** Cancel

### 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"

**Command configuration**

Task description: Publish container to AWS ECR

Disable this task

Executable: docker

Argument: push 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform

Environment variables:

Extra environment variables. e.g. JAVA\_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space.

Working sub directory:

Specify an alternative sub-directory as working directory for the task.

**How to use the Command task**

**Drag tasks here to make them final**

Add task

Save Cancel

**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"

The screenshot shows the 'Command configuration' section of a deployment pipeline. On the left, a sidebar lists various tasks: Clean working directory task, Artifact download, Command, Script, Command, Script, Command, Command, Final tasks, and Add task. The 'Command' task under 'Set tag latest for image just pushed AWS ECR' is selected and highlighted in blue. The main panel shows the 'Task description' field containing 'Set tag latest for image just pushed AWS ECR'. Below it, the 'Executable' dropdown is set to 'docker', and the 'Argument' field contains 'push 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform'. Arrows point from the task description, executable, and argument fields to their respective input boxes.

### 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)"

The screenshot shows the 'Command configuration' section of a deployment pipeline. The sidebar lists tasks: Clean working directory task, Artifact download, Command, Script, Command, Command, Final tasks, and Add task. The 'Command' task under 'Remove release-image (Clean up)' is selected and highlighted in blue. The main panel shows the 'Task description' field containing 'Remove release-image (Clean up)'. Below it, the 'Executable' dropdown is set to 'docker', and the 'Argument' field contains 'rmi vcp-netcore-webappdem01:release'. Arrows point from the task description, executable, and argument fields to their respective input boxes.

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"

Environment: DEV

How you want to deploy

Tasks define the steps involved in deploying a release to the environment.

Edit tasks

Other environment settings

These settings are not strictly necessary for your deployment to run, but they can be very helpful and allow you to make Bamboo deployments go just right.

Triggers 1 Agents assignment Notifications 1 Variables 2 Environment permissions

**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"

After successful build plan  
master CI to DEV

Add trigger

Trigger configuration

Trigger description  
master CI to DEV

Disable this trigger

Branch to trigger this deployment

Use a custom plan branch

Use the main plan branch

Plan branch master

Save trigger Cancel

## 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:

Edit environment notifications: DEV [How environment notifications work](#)

Define who receives notifications about deployment events for this environment.

**Add notification**

| Event             | Notification recipient | Actions |
|-------------------|------------------------|---------|
| Deployment failed | David Sorbona (user)   |         |

There is currently no instant messaging server configured for Bamboo. Instant message notifications will not be sent.

## Environment variables

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

Edit variables: DEV [How deployment variables work](#)

Define variables to be used by your deployment tasks. You can use any of the keys below in your task configuration or scripts and it will be automatically substituted with the value. You can override global variables by using the same key.

For task configuration fields, use the syntax \${bamboo.myvariablename}. For inline scripts, variables are exposed as shell environment variables which can be accessed using the syntax \$bamboo\_MY\_VARIABLE\_NAME (Linux/Mac OS X) or %BAMBOO\_MY\_VARIABLE\_NAME% (Windows).

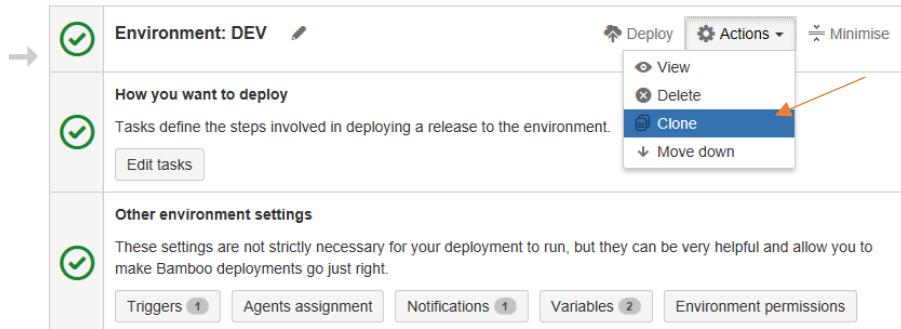
| Variable name    | Value                | Action |
|------------------|----------------------|--------|
| awsAccountNumber | 310827469077         |        |
| awsECSRole       | BambooAccessFromCore |        |

- awsAccountNumber 310827469077
- awsECSRole BambooAccessFromCore

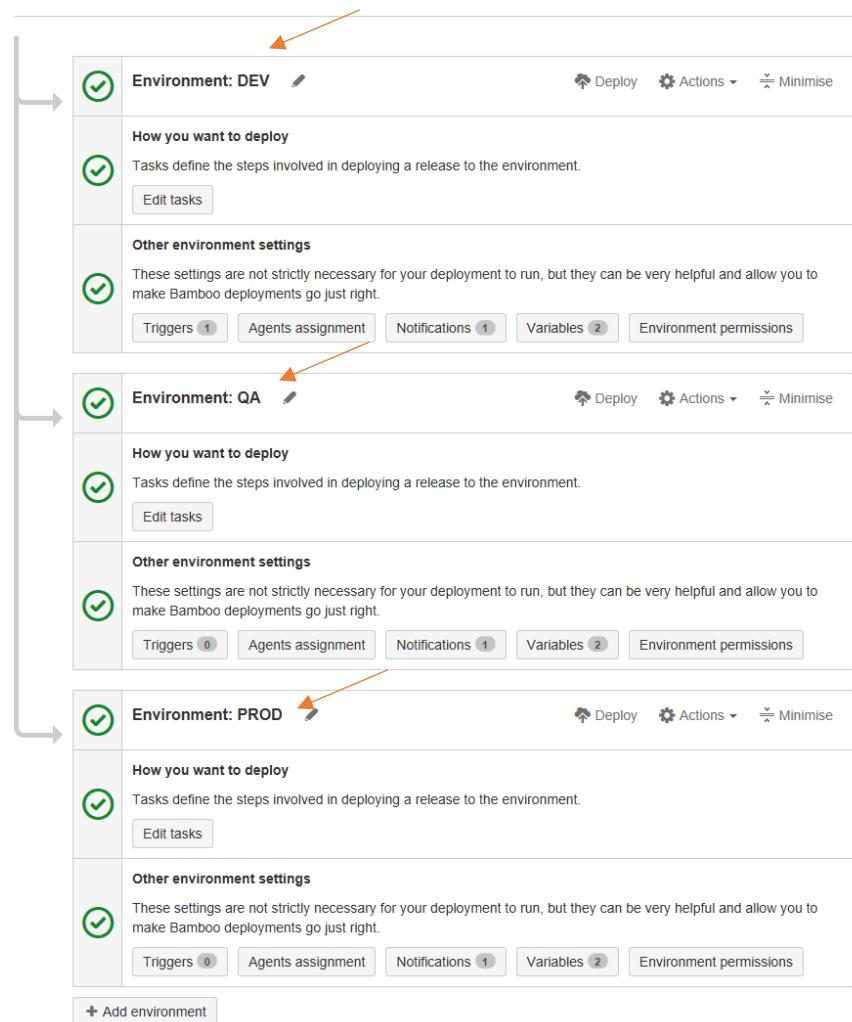
NOTE: This information is provided by the MCS team.

## 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.

Deployment projects / Viacom content platform - vcp-netcore-webappdem01-master  
**Configuration: Viacom content platform - vcp-netcore-webappdem01-master**  
CI/CD demo plan part of the Viacom content platform continuous delivery pipeline model for master branch

What you want to deploy

Source build plan Viacom content platform › vcp-netcore-webappdem01  
Available artifacts release-image-artifact

Edit build plan Release versioning Project permissions

Environment: DEV Deploy Edit...  
Environment: QA Deploy Edit...  
Environment: PROD Deploy Edit...

Then set the permission like this:

|                              | View                                | Edit                                |
|------------------------------|-------------------------------------|-------------------------------------|
| Users                        | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| David Sorbona                | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Groups                       | <input checked="" type="checkbox"/> | <input type="checkbox"/>            |
| content_platform_engineering | <input checked="" type="checkbox"/> | <input type="checkbox"/>            |
| Other                        | <input checked="" type="checkbox"/> | <input type="checkbox"/>            |
| Logged in users              | <input checked="" type="checkbox"/> | <input type="checkbox"/>            |
| Anonymous users              | <input checked="" type="checkbox"/> |                                     |

## 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.

Update tasks: DEV How deployment tasks work

What tasks need to happen to make this deployment a success 31 agents have the capabilities to deploy this environment

|  |   |   |  |  |   |  |
|--|---|---|--|--|---|--|
| <code>Clean working directory task</code><br>Clean working directory | <code>Artifact download</code><br>Download release contents | <code>Command</code><br>Load release-image from artifacts | <code>Script</code><br>Create directory for AWS resource definition repository | <code>Script</code><br>Get AWS ECR Login info and export variables | <code>Docker</code><br>Pull git-helper container-image from AWS ECR | <code>Docker</code><br>Clone aws-task-definitions repository thru git-helper docker tool |
|--|---|---|--|--|---|--|

**Script configuration** How to use the Script task

Task description: `Tag container image for AWS ECR`

Disable this task

Interpreter: `Shell`

An interpreter is chosen based on the shebang line of your script.

Script location: `Inline`

Script body\*:

```
1 azonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:${bamboo.dep
2 azonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:dev-latest)
3
```



### 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”

**Command configuration**

Task description  
Set tag latest for image just pushed AWS ECR

Disable this task

Executable  
docker

Argument  
amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:dev-latest

Environment variables

Extra environment variables. e.g. JAVA\_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space.

Working sub directory

Specify an alternative sub-directory as working directory for the task.

**Save**   [Cancel](#)

How to use the Command task

**Step List:**

- Clean working directory task
- Artifact download
- Command
- Script
- Script
- Docker
- Docker
- Script
- Script
- Command
- Command
- Docker

The "Argument" field is highlighted with a red box and an arrow points to it from the left.

### 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?)

|   |      |                         |      |                      |  |
|---|------|-------------------------|------|----------------------|--|
| Viacom content platform - vcp-netcore-webappdem01-development | DEV  | dev-build-24.release-25 | Logs | 30 Mar 2017 07:03 AM |  |
| Viacom content platform - vcp-netcore-webappdem01-master      | DEV  | build-62.release-19     | Logs | 30 Mar 2017 07:03 AM |  |
|   | QA   |                         |      | Never deployed       |  |
|   | PROD |                         |      | Never deployed       |  |

## 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

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

The screenshot shows the deployment project summary for the "Viacom content platform - vcp-netcore-webappdem01-master" project. It lists three environments: DEV, QA, and PROD. The DEV environment has a deployment entry for "build-66.release-23". The QA and PROD environments have no deployed artifacts. The "Actions" column for each environment contains icons for Deploy, Edit, and Delete.

| Environment | Release                             | Result         | Completed            | Actions |
|-------------|-------------------------------------|----------------|----------------------|---------|
| DEV         | <a href="#">build-66.release-23</a> | Logs           | 03 Apr 2017 07:04 AM |         |
| QA          |                                     | Never deployed |                      |         |
| PROD        |                                     | Never deployed |                      |         |

- Flag the deployment as Approved or Broken as needed

The screenshot shows the release details for "build-66.release-23". It includes tabs for Status, Commits (0), Issues (0), and Variables. The "Status" tab is active, showing deployment results for environments DEV, QA, and PROD. The DEV environment is marked as "SUCCESS". The "Actions" column for each environment contains icons for Deploy, Edit, and Delete. To the right, there is a panel for "build-66.release-23 details" which shows the creation time, deployment project, artifacts provided by, and release contents.

| Environment | Status  | Deployment result   | Completed            | Trigger             | Actions |
|-------------|---|---|----------------------|---------------------|---------|
| DEV         | <span style="background-color: green; color: white; padding: 2px;">SUCCESS</span> | <span style="background-color: green; color: white; padding: 2px;">SUCCESS</span> | 03 Apr 2017 06:04 AM | Child of VCP-BAC-66 |         |
| QA          | Never deployed  |   |                      |                     |         |
| PROD        | Never deployed  |   |                      |                     |         |

Created 6 hours ago

Deployment project Viacom content platform - vcp-netcore-webappdem01-master

Artifacts provided by #66 Viacom content platform > vcp-netcore-webappdem01

Release contents release-image-artifact

## Rollback deployment

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

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

Deployment projects  
Viacom content platform - vcp-netcore-webappdem01-master  
CI/CD demo plan part of the Viacom content platform continuous delivery pipeline model for master branch

**Project summary**   [Releases](#)

Deployment project summary

Source build plan [Viacom content platform > vcp-netcore-webappdem01](#)

Available artifacts [release-image-artifact](#)

| Environment | Release                             | Result         | Completed            | Actions |
|-------------|-------------------------------------|----------------|----------------------|---------|
| DEV         | <a href="#">build-66.release-23</a> | Logs           | 03 Apr 2017 07:04 AM |         |
| QA          |                                     | Never deployed |                      |         |
| PROD        |                                     | Never deployed |                      |         |

- Rollback to the release you need:

Deployment projects / Viacom content platform - vcp-netcore-webappdem01-master  
Environment: DEV  
CI/CD demo plan part of the Viacom content platform continuous delivery pipeline model for master branch

**Current status**

Release [build-66.release-23](#)  
 master

Result Logs  
Completed 03 Apr 2017 06:04 AM  
Trigger Child of [VCP-BAC-66](#)

**Previously deployed releases**

All releases that used to be deployed on this environment, descending in chronological order.

| Release                             | Result | Completed            | Trigger                             | Actions |
|-------------------------------------|--------|----------------------|-------------------------------------|---------|
| <a href="#">build-65.release-22</a> | Logs   | 02 Apr 2017 06:02 AM | Child of <a href="#">VCP-BAC-65</a> |         |
| <a href="#">build-64.release-21</a> | Logs   | 01 Apr 2017 06:03 AM | Child of <a href="#">VCP-BAC-64</a> |         |
| <a href="#">build-63.release-20</a> | Logs   | 31 Mar 2017 06:03 AM | Child of <a href="#">VCP-BAC-63</a> |         |

## 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:

Deployment projects  
**Viacom content platform - vcp-netcore-webappdem01-master**  
CI/CD demo plan part of the Viacom content platform continuous delivery pipeline model for master branch

**Project summary**   [Releases](#)

Deployment project summary

Source build plan [Viacom content platform > vcp-netcore-webappdem01](#)

Available artifacts [release-image-artifact](#)

| Environment | Release             | Result         | Completed            | Actions |
|-------------|---------------------|----------------|----------------------|---------|
| DEV         | build-66.release-23 | Logs           | 03 Apr 2017 07:04 AM |         |
| QA          |                     | Never deployed |                      |         |
| PROD        |                     | Never deployed |                      |         |

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

Deployment preview: QA [How deployment releases work](#)

Select a release to  Create new release from build result  
 Promote existing release to this environment

deploy Release\*

Release details  
**build-66.release-23**  
This is your first deployment to this environment.

Deployment history  
 1 of 3 environments

Reviewed  
Not reviewed

Execution details  
7 tasks

[Start deployment](#) [Cancel](#)

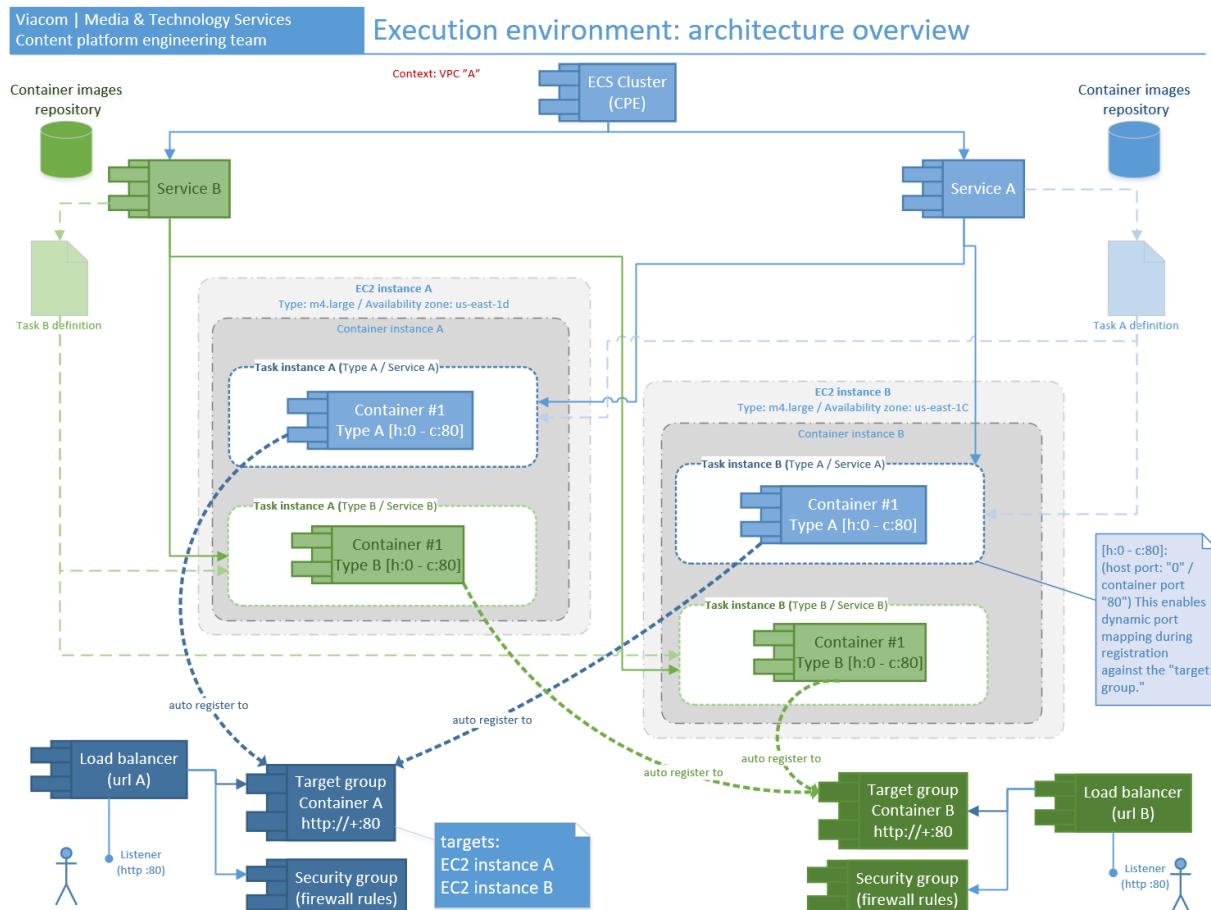
## Setup execution environment in cloud provider

### 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.

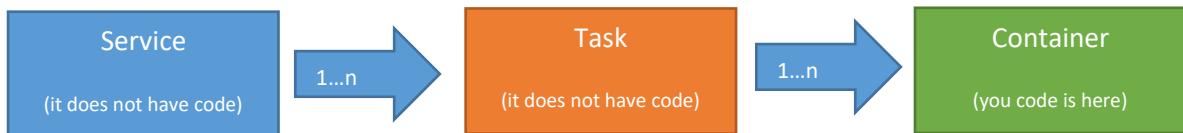


**NOTE:** There are more resources that were not included in the diagram.

**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.

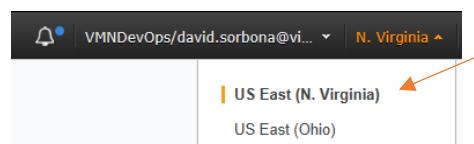
To create the task definition, follow the next steps:

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

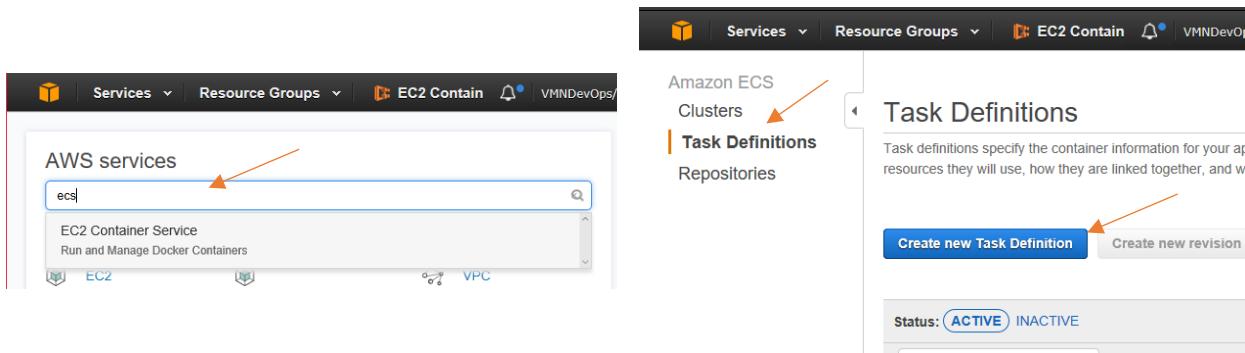
<https://fs.viacomcloud.com/adfs/ls/IdpInitiatedSignon.aspx>



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



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



4. Completes the form as shown:

The screenshot shows the 'Create a Task Definition' page in the AWS ECS console. The left sidebar has 'Clusters' (selected), 'Task Definitions' (selected), and 'Repositories'. The main area has fields for 'Task Definition Name\*' (set to 'vcp-netcore-webappdem01'), 'Task Role' (set to 'None'), and 'Network Mode' (set to 'Bridge'). Arrows point from the text descriptions to their respective input fields.

**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:

The screenshot shows the 'Container Definitions' section. It includes a 'Type' column with a '+ Add constraint' button and an 'Expression' column. Below this is a table with columns 'Container Name', 'Image', and 'Hard/Soft memory limits (MB)'. A blue arrow points to the '+ Add container' button at the bottom of the table.

6. Completes the “Standard” section as follows:

Add container

▼ Standard

**Container name\*** vcp-netcore-webappdem01

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

Custom image format: [registry-url]/[namespace]/[image]:[tag]

**Memory Limits (MB)\*** Hard limit 128

**+ Add Soft limit**

Define hard and/or soft memory limits in MiB for your container. Hard and soft limits correspond to the ‘memory’ and ‘memoryReservation’ parameters, respectively, in task definitions.  
ECS recommends 300-500 MB as a starting point for web applications.

**Port mappings**

|           |                |          |
|-----------|----------------|----------|
| Host port | Container port | Protocol |
| 0         | 80             | tcp      |

**+ Add port mapping**

▼ Advanced container configuration

**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 used 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:

▼ Advanced container configuration

**ENVIRONMENT**

|           |                                     |
|-----------|-------------------------------------|
| CPU units | 256                                 |
| Essential | <input checked="" type="checkbox"/> |

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”

#### Constraint

Constraints allow you to filter the instances used for your placement strategies using built-in or custom attributes. The scheduler first filters the instances that match the constraints and then applies the placement strategy to place the task.

| Type                             | Expression |
|----------------------------------|------------|
| <a href="#">+ Add constraint</a> |            |

**Container Definitions**

| Add container           |   |                              |              |
|-------------------------|---|------------------------------|--------------|
| Container Name          | Image   | Hard/Soft memory limits (MB) | Essential... |
| vcp-netcore-webappdem01 | 310827469077.dkr.ecr.us-east-1.amazonaws.com... | 128/-                        | true         |

**Volumes**

| Name       | Source Path |
|------------|-------------|
| No results |             |

[+ Add volume](#)

[Configure via JSON](#)

**Create** **Cancel**

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

## Task Definition: VCP-TEST:1

View detailed information for your task definition. To modify the task definition, you need to create a new revision and then make the required changes to the task definition

[Create new revision](#) [Actions ▾](#)

[Builder](#) [JSON](#)

```
{
  "requiresAttributes": [
    {
      "value": null,
      "name": "com.amazonaws.ecs.capability.ecr-auth",
      "targetId": null,
      "targetType": null
    }
  ]
}
```

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”

```
"links": null,  
"workingDirectory": null,  
"readonlyRootFilesystem": null,  
"image": "310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:latest",  
"command": null,  
"user": null,  
"dockerLabels": null,  
"memory": null,
```

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

```
"links": null,  
"workingDirectory": null,  
"readonlyRootFilesystem": null,  
"image": "310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-netcore-webappdem01:[CONTAINER-TAG]",  
"command": [],  
"user": null,  
"dockerLabels": null
```

### NOTE

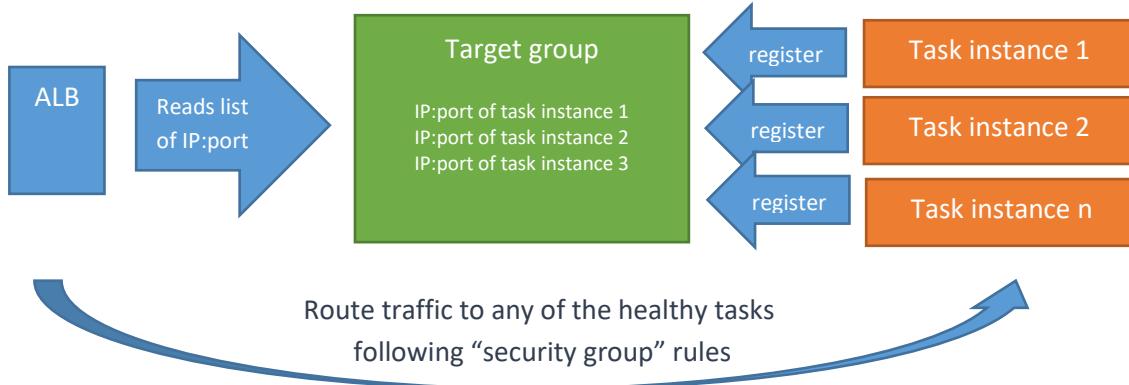
The same technique could be used to update other variables on deploy time, like the amount of memory, CPU or any other task/container property.

**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.

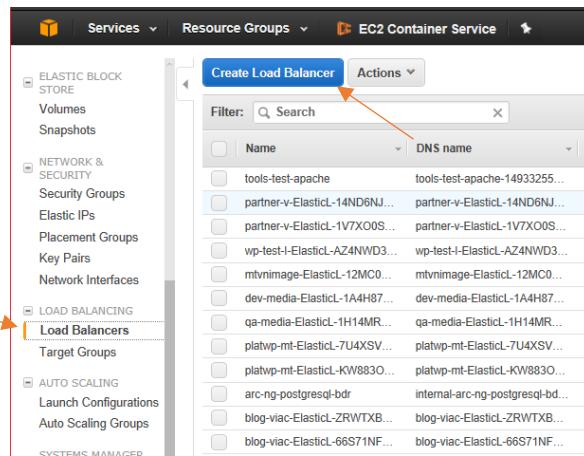
**CRITICAL:** Make sure you create all resources in the same VPC ID

To create the AWS Load balancer, follow this steps:

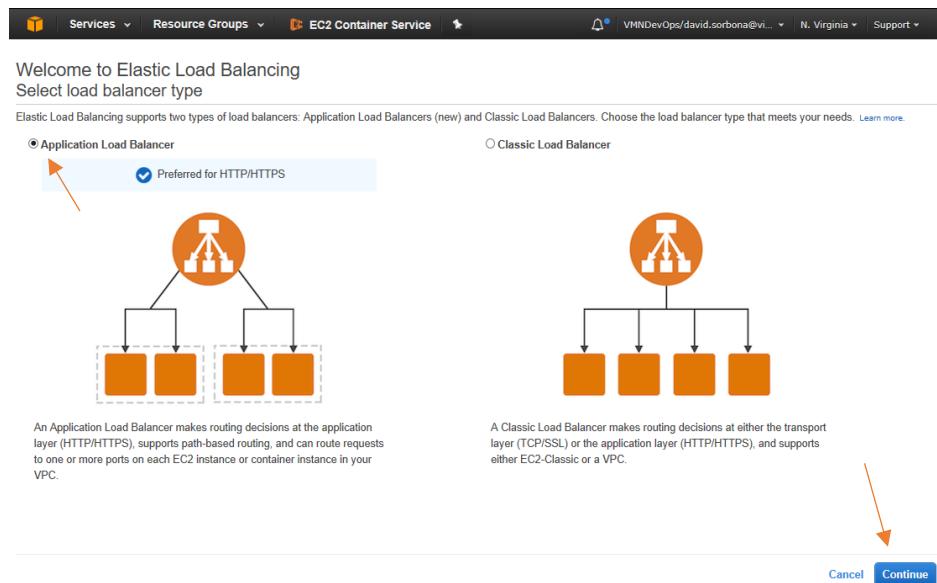
1. In the AWS web portal go to the EC2 dashboard for us-east-1 region:

<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”

#### Step 1: Configure Load Balancer

##### Basic Configuration

To configure your load balancer, provide a name, select a scheme, specify one or more listeners, and select a network. The default configuration is an Internet-facing load balancer in the selected network with a listener that receives HTTP traffic on port 80.

|                 |   |
|-----------------|---|
| Name            | <input type="text" value="CPE-vcp-netcore-webappdem01"/>                        |
| Scheme          | <input checked="" type="radio"/> Internet-facing <input type="radio"/> Internal |
| IP address type | <input type="text" value="ipv4"/>   |

##### Listeners

A listener is a process that checks for connection requests, using the protocol and port that you configured.

| Load Balancer Protocol            | Load Balancer Port              |
|-----------------------------------|---------------------------------|
| <input type="text" value="HTTP"/> | <input type="text" value="80"/> |
| <a href="#">Add listener</a>      |                                 |

##### Availability Zones

Specify the Availability Zones to enable for your load balancer. The load balancer routes traffic to the targets in these Availability Zones only. You can specify only one subnet per Availability Zone. You must specify subnets from at least two Availability Zones to increase the availability of your load balancer.

| VPC  | vpc-55a90631 (10.242.88.0/21)   underlay-0-vpc |                  |                           |                                  |
|--|--|------------------|---------------------------|----------------------------------|
| Availability Zone                              | Subnet ID                                      | Subnet IPv4 CIDR | Name                      |                                  |
| <input checked="" type="checkbox"/> us-east-1c | subnet-dec7f3f5                                | 10.242.92.0/24   | underlay-0-public-subnet1 | <a href="#">Change subnet...</a> |
| <input checked="" type="checkbox"/> us-east-1d | subnet-369d7040                                | 10.242.93.0/24   | underlay-0-public-subnet2 | <a href="#">Change subnet...</a> |

##### Tags

[Cancel](#) [Next: Configure Security Settings](#)

- **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”

#### Step 2: Configure Security Settings

**⚠ Improve your load balancer's security. Your load balancer is not using any secure listener.**  
If your traffic to the load balancer needs to be secure, use the HTTPS protocol for your front-end connection. You can go back to the first step to add/configure secure listeners under [Basic Configuration](#) section.  
You can also continue with current settings.

[Cancel](#) [Previous](#) [Next: Configure Security Groups](#)

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

#### Step 3: Configure Security Groups

A security group is a set of firewall rules that control the traffic to your load balancer. On this page, you can add rules to allow specific traffic to reach your load balancer. First, decide whether to create a new security group or select an existing one.

**Assign a security group:**  Create a new security group  
 Select an existing security group

**Security group name:**

**Description:**

| Type            | Protocol | Port Range | Source             |
|-----------------|----------|------------|--------------------|
| Custom TCP Rule | TCP      | 80         | Custom 0.0.0.0/:/0 |

[Add Rule](#)

[Cancel](#) [Previous](#) [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"

#### Step 4: Configure Routing

Your load balancer routes requests to the targets in this target group using the protocol and port that you specify, and performs health checks on the targets using these health check settings. Note that each target group can be associated with only one load balancer.

##### Target group

|              |   |   |
|--------------|---|---|
| Target group | <input type="text" value="New target group"/>             | ← |
| Name         | <input type="text" value="CPE-webappdem01-target-group"/> | ← |
| Protocol     | <input type="text" value="HTTP"/>                         | ← |
| Port         | <input type="text" value="80"/>                           | ← |

|          |                                   |   |
|----------|-----------------------------------|---|
| Protocol | <input type="text" value="HTTP"/> | ← |
| Path     | <input type="text" value="/"/>    | ← |

▼ Advanced health check settings

|                     |   |                                |
|---------------------|---|--------------------------------|
| Port                | <input checked="" type="radio"/> traffic port | <input type="radio"/> override |
| Healthy threshold   | <input type="text" value="5"/>                |                                |
| Unhealthy threshold | <input type="text" value="2"/>                |                                |
| Timeout             | <input type="text" value="5"/>                | seconds                        |
| Interval            | <input type="text" value="30"/>               | seconds                        |
| Success codes       | <input type="text" value="200"/>              |                                |

Cancel
Previous
Next: Register Targets

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

#### Step 5: Register Targets

Register targets with your target group. If you register an instance running in an enabled Availability Zone, the load balancer starts routing requests to the instance as soon as the registration process completes and the instance passes the initial health checks.

##### Registered instances

To deregister instances, select one or more registered instances and then click Remove.

| <input type="button" value="Remove"/> | Instance | Name | Port | State | Security groups | Zone |
|---------------------------------------|----------|------|------|-------|-----------------|------|
| No instances available.               |          |      |      |       |                 |      |

##### Instances

To register additional instances, select one or more running instances, specify a port, and then click Add. The default port is the port specified for the target group. If the instance is already registered on the specified port, you must specify a different port.

| <input type="button" value="Add to registered"/> | on port                | 80   |
|--|------------------------|--|
| <input type="text" value="Search Instances"/>    |                        |  |
| <input type="checkbox"/> Instance                | Name                   | State  |
| <input type="checkbox"/> i-03144ef05f59e7bd2     | CPE-Non-Productio...   | <span style="color: green;">● running</span> |
| <input type="checkbox"/> i-0b57518331e3642...    | blog.viacom.com-d...   | <span style="color: green;">● running</span> |
| <input type="checkbox"/> i-038275291aeffd71e     | CPE-Non-Productio...   | <span style="color: green;">● running</span> |
| <input type="checkbox"/> i-049278e3cea7483...    | toolteam-jenkins-g...  | <span style="color: green;">● running</span> |
| <input type="checkbox"/> i-03ce267d4a14c0ebb     | tools-test-ELBAuto...  | <span style="color: green;">● running</span> |
| <input type="checkbox"/> i-0b35c83117560a9...    | dss-04 (dss.mtv.com)   | <span style="color: green;">● running</span> |
| <input type="checkbox"/> i-0a1b78a3c46669f09     | vms-search-dev-EL...   | <span style="color: green;">● running</span> |
| <input type="checkbox"/> i-04e66ca2a658583...    | qa-awsplatform-01...   | <span style="color: green;">● running</span> |
| <input type="checkbox"/> i-0fde5c12ff0f9e151     | dev-platform-03        | <span style="color: green;">● running</span> |
| <input type="checkbox"/> i-017c56bbc5b7993c1     | toolteam-platform:j... | <span style="color: green;">● running</span> |

Cancel
Previous
Next: Review



9. Review the configuration and click on “Create”

**Step 6: Review**  
Please review the load balancer details before continuing

**Load balancer**

|                 |   |      |
|-----------------|---|------|
| Name            | CPE-vcp-netcore-webappdem01   | Edit |
| Scheme          | internet-facing   |      |
| Listeners       | Port:80 - Protocol:HTTP   |      |
| IP address type | ipv4  |      |
| VPC             | vpc-55a90631 (underlay-0-vpc)   |      |
| Subnets         | subnet-dec7f5f (underlay-0-public-subnet1), subnet-369d7040 (underlay-0-public-subnet2) |      |
| Tags            |   |      |

**Security settings**

|                      |  |      |
|----------------------|--|------|
| Certificate name     |  | Edit |
| Security policy name |  |      |

**Security groups**

|                 |                             |      |
|-----------------|-----------------------------|------|
| Security groups | CPE-vcp-netcore-webappdem01 | Edit |
|-----------------|-----------------------------|------|

**Routing**

|                       |                              |  |
|-----------------------|------------------------------|--|
| Target group          | New target group             |  |
| Target group name     | CPE-webappdem01-target-group |  |
| Port                  | 80                           |  |
| Protocol              | HTTP                         |  |
| Health check protocol | HTTP                         |  |
| Path                  | /                            |  |
| Health check port     | traffic port                 |  |
| Healthy threshold     | 5                            |  |
| Unhealthy threshold   | 2                            |  |
| Timeout               | 5                            |  |
| Interval              | 30                           |  |
| Success codes         | 200                          |  |

**Targets**

|           |  |      |
|-----------|--|------|
| Instances |  | Edit |
|-----------|--|------|

Cancel Previous Create

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

Load Balancer Creation Status

✓ Successfully created load balancer  
Load balancer CPE-vcp-netcore-webappdem01 was successfully created.  
Note: It might take a few minutes for your load balancer to be fully set up and ready to route traffic, and for the targets to complete the registration process and pass the initial health checks.

Close

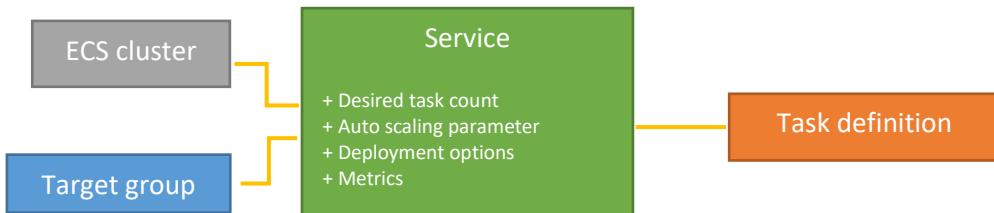
Note that the provisioning of this could take several minutes. When its state is “Active” it is ready to rock!

| Name                        | DNS name                    | State        | VPC ID       |
|-----------------------------|-----------------------------|--------------|--------------|
| CPE-vcp-netcore-webappdem01 | CPE-vcp-netcore-webappdem01 | provisioning | vpc-55a90631 |

**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”

### Create Service

A service lets you specify how many copies of your task definition to run and maintain in a cluster. You can optionally use an Elastic Load Balancing load balancer to distribute incoming traffic to containers in your service. Amazon ECS maintains that number of tasks and coordinates task scheduling with the load balancer. You can also optionally use Service Auto Scaling to adjust the number of tasks in your service.

Task Definition: vcp-netcore-webappdem01

Cluster: CPE-Non-Production-ALB-Cluster-ECSCluster-17JM3IP1...

Service name: vcp-netcore-webappdem01-dev

Number of tasks: 4

Minimum healthy percent: 25

Maximum percent: 200

### Task Placement

Lets you customize how tasks are placed on instances within your cluster. Different placement strategies are available to optimize for availability and efficiency.

Placement Templates: AZ Balanced Spread Edit

This template will spread tasks across availability zones and within the availability zone spread tasks across instances. [Learn more.](#)

Strategy: spread(attribute:ecs.availability-zone), spread(instanceId)

### Optional configurations

#### Elastic load balancing

Connect an Elastic Load Balancing load balancer to distribute traffic to each task in your service from a single DNS endpoint.

[Configure ELB](#)

#### Service Auto scaling

Automatically adjust your service's desired count up and down in response to CloudWatch alarms.

[Configure Service Auto Scaling](#)

[Cancel](#) [Create Service](#)

- 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 your service)

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

### Elastic Load Balancing (optional)

An Elastic Load Balancing load balancer distributes incoming traffic across the tasks running in your service. Choose an existing load balancer, or create a new one in the [Amazon EC2 console](#). When you are finished configuring your load balancer, choose **Save** to continue.

ELB type:

- None  
Your service will not use a load balancer.
- Application Load Balancer  
Allows containers to use dynamic host port mapping (multiple tasks allowed per container instance). Multiple services can use the same listener port on a single load balancer with rule-based routing and paths.
- Classic Load Balancer  
Requires static host port mappings (only one task allowed per container instance); rule-based routing and paths are not supported.

Select IAM role for service: ecsServiceRole

ELB Name: CPE-vcp-netcore-webappdem01

Container to load balance

Select a Container: vcp-netcore-webappdem01:80

Add to ELB

Back Save

- **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”

Container to load balance

vcp-netcore-webappdem01 : 80

Remove X

Listener port: 80:HTTP

Listener protocol: HTTP

Target group name: CPE-webappdem01-target...

Target group protocol: HTTP

Path pattern: /

Evaluation order: default

Health check path: /

Additional health check options can be configured in the ELB console after you create your service.

Back Save



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

### Task Placement

Lets you customize how tasks are placed on instances within your cluster. Different placement strategies are available to optimize for availability and efficiency.

Placement Templates

AZ Balanced Spread

This template will spread tasks across availability zones and within the availability zone spread tasks across instances. [Learn more](#).

Strategy: spread(attribute:ecs.availability-zone), spread(instanceId)

### Optional configurations

#### Elastic load balancing

Connect an Elastic Load Balancing load balancer to distribute traffic to each task in your service from a single DNS endpoint.

[Configure ELB](#)

#### Service Auto scaling

Automatically adjust your service's desired count up and down in response to CloudWatch alarms.

[Configure Service Auto Scaling](#)

Container Name: vcp-netcore-webappdem01

Container Port: 80

ELB Name: CPE-vcp-netcore-webappdem01

Target Group: CPE-webappdem01-target-group

Health Check Path: /

Listener Port: 80

Path-pattern: /

Service Role: ecsServiceRole

[Cancel](#) [Create Service](#)

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

### Launch Status

ECS Service status - 2 of 2 completed

#### Create Load Balancer

#### IAM Policy

IAM Service policy attached

IAM Service policy attached to the role: arn:aws:iam::310827469077:role/ecsServiceRole. View policy: [AmazonEC2ContainerServiceRole](#)

#### Create Service

Create service: vcp-netcore-webappdem01-dev

Service created

Service created. Tasks will start momentarily. View: [vcp-netcore-webappdem01-dev](#)

[Back](#) [View Service](#)

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:

| Target Group Name            | Container Name          | Container Port |
|------------------------------|-------------------------|----------------|
| CPE-webappdem01-target-group | vcp-netcore-webappdem01 | 80             |

**Deployment Options**

Minimum healthy percent: 25 ⓘ  
Maximum percent: 200 ⓘ

**Task Placement**

Strategy: spread(attribute.ecs.availability-zone), spread(instanceid)  
Constraint: No constraints

**Tasks** | Events | Deployments | Auto Scaling | Metrics

Last updated on April 18, 2017 8:52:09 PM (0m ago)

| Task                                 | Task Definition             | Group                               | Last status | Desired status |
|--------------------------------------|-----------------------------|-------------------------------------|-------------|----------------|
| 37957fc0-d233-4192-b7cb-3cde0f29a3a0 | vcp-netcore-webappdem01:115 | service.vcp-netcore-webappdem01-dev | RUNNING     | RUNNING        |
| 401543ed-66fd-4e16-a745-bf5d9fe734c5 | vcp-netcore-webappdem01:115 | service.vcp-netcore-webappdem01-dev | RUNNING     | RUNNING        |
| ad9b9790-2ce9-49b7-93ab-e6e97b7b6d9e | vcp-netcore-webappdem01:115 | service.vcp-netcore-webappdem01-dev | RUNNING     | RUNNING        |
| d5ae9d96-6e32-49e5-b1fd-a9dc6ea3417d | vcp-netcore-webappdem01:115 | service.vcp-netcore-webappdem01-dev | RUNNING     | RUNNING        |

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”

| Target Group Name            | Container Name          | Container Port |
|------------------------------|-------------------------|----------------|
| CPE-webappdem01-target-group | vcp-netcore-webappdem01 | 80             |

**Deployment Options**

## 2. Click on “Load balancer”

Target group: CPE-webappdem01-target-group

Description Targets Health checks Monitoring Tags

### Basic Configuration

|               |   |   |
|---------------|---|---|
| Name          | i | CPE-webappdem01-target-group  |
| ARN           | i | arn:aws:elasticloadbalancing:us-east-1:31062746group/e343e214e20574ca |
| Protocol      | i | HTTP  |
| Port          | i | 80  |
| VPC           | i | vpc-55a90631  |
| Load balancer | i | CPE-vcp-netcore-webappdem01   |

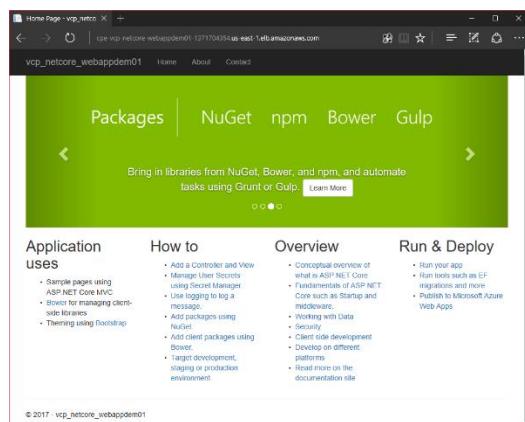
3. Copy the DNS name and use it as URL in your favorite browser:

| Basic Configuration |   |
|---------------------|---|
| Description         | CPE-vcp-netcore-webappdem01   |
| Listeners           |   |
| Monitoring          |   |
| Tags                |   |
| <b>DNS name:</b>    | CPE-vcp-netcore-webappdem01-1371704354.us-east-1.elb.amazonaws.com (A Record) |
| <b>Scheme:</b>      | internet-facing   |

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

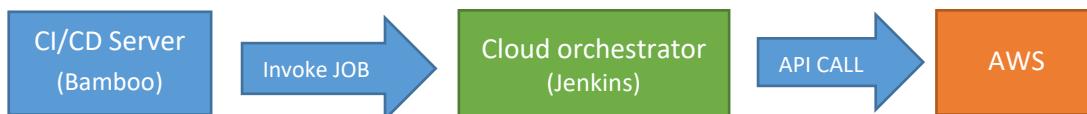
If you've followed this procedure in order, by now you have created three things:

- 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

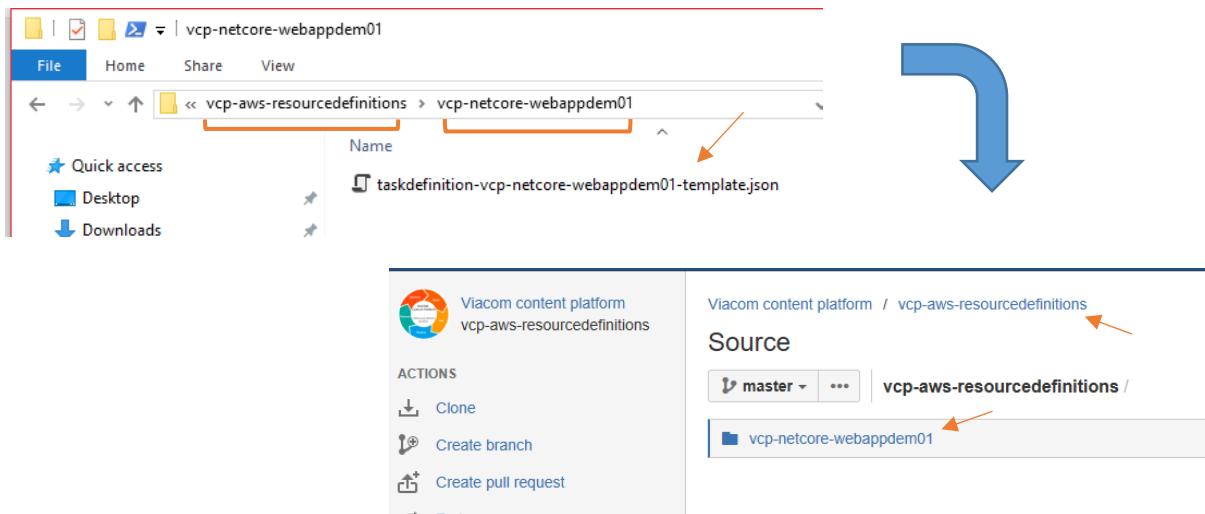
The new task definition json file will be stored in a repository in the Viacom Bitbucket instance called “vcp-aws-resourcedefinitions”. I will not explain how to do this because it is easy and because we’ve already explained how to create a repository in Bitbucket earlier. Anyway the repository has already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

<https://stash.mtv.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 update 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.mtv.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)

The screenshot shows the Bamboo deployment interface for the 'vcp-netcore-webappdem01-master' project. The 'Edit task' page is displayed for the 'Environment: DEV'. The 'Edit tasks' button is highlighted with a red arrow. Other visible sections include 'Source build plan' (Viacom content platform > vcp-netcore-webappdem01), 'Available artifacts' (release-image-artifact), and 'Other environment settings'.

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)

Update tasks: DEV How deployment tasks work

What tasks need to happen to make this deployment a success 31 agents have the capabilities to deploy this environment

|   |  |
|---|--|
| <div style="border-bottom: 1px solid #ccc; padding-bottom: 5px;"> <span style="font-size: 10px;">Clean working directory task</span><br/> <span style="font-size: 10px;">Clean working directory</span> </div> <div style="border-bottom: 1px solid #ccc; padding-bottom: 5px;"> <span style="font-size: 10px;">Artifact download</span><br/> <span style="font-size: 10px;">Download release contents</span> </div> <div style="border-bottom: 1px solid #ccc; padding-bottom: 5px;"> <span style="font-size: 10px;">Command</span><br/> <span style="font-size: 10px;">Load release-image from artifacts</span> </div> <div style="background-color: #0070C0; color: white; padding: 2px 5px; margin-bottom: 5px;"> <span style="font-size: 10px;">Script</span><br/> <span style="font-size: 10px;">Create directory for AWS resource definition repository</span> </div> <div style="border-bottom: 1px solid #ccc; padding-bottom: 5px;"> <span style="font-size: 10px;">Script</span><br/> <span style="font-size: 10px;">Get AWS ECR Login info and export variables</span> </div> <div style="border-bottom: 1px solid #ccc; padding-bottom: 5px;"> <span style="font-size: 10px;">Script</span><br/> <span style="font-size: 10px;">Tag container image for AWS ECR</span> </div> <div style="border-bottom: 1px solid #ccc; padding-bottom: 5px;"> <span style="font-size: 10px;">Command</span><br/> <span style="font-size: 10px;">Publish container to AWS ECR</span> </div> <div style="border-bottom: 1px solid #ccc; padding-bottom: 5px;"> <span style="font-size: 10px;">Command</span><br/> <span style="font-size: 10px;">Set tag latest for image just pushed AWS ECR</span> </div> <div style="border-bottom: 1px solid #ccc; padding-bottom: 5px;"> <span style="font-size: 10px;"><b>Final tasks</b> Are always executed even if a previous task fails</span> </div> <div style="border-bottom: 1px solid #ccc; padding-bottom: 5px;"> <span style="font-size: 10px;">Command</span><br/> <span style="font-size: 10px;">Remove release-image (Clean up)</span> </div> <div style="border-bottom: 1px solid #ccc; padding-bottom: 5px; text-align: center;"> <span style="border: 1px solid #ccc; padding: 2px 10px; background-color: #f0f0f0;">Add task</span> </div> | <p><b>Script configuration</b> <span style="float: right;">How to use the Script task</span></p> <p>Task description<br/><input type="text" value="Create directory for AWS resource definition repository"/></p> <p><input type="checkbox"/> Disable this task</p> <p>Interpreter<br/><input type="text" value="/bin/sh or cmd.exe"/></p> <p>Run your script with /bin/sh or cmd.exe</p> <p>Script location<br/><input type="text" value="Inline"/></p> <p>Script body*<br/> <pre style="background-color: #e0e0ff; padding: 5px;">1 #!/bin/bash 2 mkdir ./vcp-aws-resourcedefinitions 3 4 5</pre> </p> <p>Argument<br/><input type="text"/></p> <p>Environment variables<br/><input type="text"/></p> <p>Working sub directory<br/><input type="text"/></p> <p style="text-align: center; margin-top: 10px;"> <span style="background-color: #0070C0; color: white; border: 1px solid #0070C0; padding: 2px 10px; border-radius: 5px;">Save</span> <span style="margin-left: 10px;">Cancel</span> </p> |
|---|--|

#### 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)

Update tasks: DEV How deployment tasks work

What tasks need to happen to make this deployment a success 31 agents have the capabilities to deploy this environment

|  |  |
|--|--|
| <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em;">Clean working directory task</span><br/> <span style="font-size: 0.8em;">Clean working directory</span> </div> <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em;">Artifact download</span><br/> <span style="font-size: 0.8em;">Download release contents</span> </div> <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em;">Command</span><br/> <span style="font-size: 0.8em;">Load release-image from artifacts</span> </div> <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em;">Script</span><br/> <span style="font-size: 0.8em;">Create directory for AWS resource definition repository</span> </div> <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em;">Script</span><br/> <span style="font-size: 0.8em;">Get AWS ECR Login Info and export variables</span> </div> <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em; background-color: #0070C0; color: white; padding: 2px 5px;">Docker</span><br/> <span style="font-size: 0.8em;">Pull git-helper container-image from AWS ECR</span> </div> <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em;">Script</span><br/> <span style="font-size: 0.8em;">Tag container image for AWS ECR</span> </div> <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em;">Command</span><br/> <span style="font-size: 0.8em;">Publish container to AWS ECR</span> </div> <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em;">Command</span><br/> <span style="font-size: 0.8em;">Set tag latest for image just pushed AWS ECR</span> </div> <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em;">Final tasks</span> <small>Are always executed even if a previous task fails</small> </div> <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em;">Command</span><br/> <span style="font-size: 0.8em;">Remove release-image (Clean up)</span> </div> <div style="border-bottom: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="font-size: 1em;">Add task</span> </div> | <p><b>Docker configuration</b> <span style="float: right;">How to use the Docker task</span></p> <p>Task description<br/> <input type="text" value="Pull git-helper container-image from AWS ECR"/></p> <p><input type="checkbox"/> Disable this task</p> <p>Command<br/> <input type="text" value="Pull a Docker image from a Docker registry"/></p> <p>The Docker command to execute</p> <p>Registry<br/> <input type="radio"/> Docker Hub<br/> <input checked="" type="radio"/> Custom registry</p> <p>Repository*<br/> <input type="text" value="310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-"/></p> <p>Registry address, repository name and optionally a tag to pull from the custom registry (e.g. 'registry.address:port(namespace/repository:tag)')</p> <p><b>Credentials</b></p> <p>Leave these credentials empty to use the agent's <code>~/.dockercfg</code> credentials.</p> <p>Username<br/> <input type="text"/></p> <p>Password<br/> <input type="password"/></p> <p>Email<br/> <input type="text"/></p> <p><b>Advanced options</b></p> <p><span style="float: right;"><input type="button" value="Save"/> <input type="button" value="Cancel"/></span></p> |
|--|--|

- **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)

The screenshot shows the 'Update tasks: DEV' screen in Bamboo. On the left, a sidebar lists various task types: Clean working directory task, Artifact download, Command, Script, Docker, Docker (selected), Script, Command, Command, Final tasks, and Command. The 'Docker' task is currently selected. The main panel displays the configuration for this task:

- Docker configuration:**
  - Task description:** Clone aws-task-definitions repository thru git-helper docker tool
  - Command:** Run a Docker container
  - Docker image:** 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-toolbox-docker-githelper:latest
  - Container environment variables:** (empty)
  - Container command:** git clone ssh://git@stash.mtv.com/vcp/vcp-aws-resourcedefinitions.git
  - Container working directory:** /
  - Additional arguments:** (empty)
  - Additional Docker run arguments:** (empty)
- Volumes:** Host directory \${bamboo.working.directory} mapped to Container data volume /data.
- Advanced options:** (checkboxes for Disable task, Detach container, Link to detached containers, and Container environment variables are shown but empty.)

At the bottom right of the configuration panel are 'Save' and 'Cancel' buttons.

- **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.mtv.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)

The screenshot shows the 'Update tasks: DEV' interface. On the left, a list of tasks is displayed, with the 'Script' task selected. The right side shows the 'Script configuration' details. The 'Interpreter' is set to '/bin/sh or cmd.exe'. The 'Script location' is 'Inline'. The 'Script body' contains the following code:

```

1  #!/bin/bash
2  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

```

At the bottom, there are 'Save' and 'Cancel' buttons.

- 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.mtv.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 screenshot shows a Bitbucket interface. At the top, there's a navigation bar with 'Bitbucket' logo, 'Projects', 'Repositories', and 'Pull requests'. Below the navigation bar, the path 'Viacom content platform / vcp-aws-resourcedefinitions' is visible. On the left, there's a sidebar with various icons for file operations like download, copy, move, etc. The main area is titled 'Source' and shows a list of files under the folder 'vcp-aws-resourcedefinitions / vcp-netcore-webappdem01 /'. Two red arrows point from the text above to the folder name 'vcp-netcore-webappdem01 /' and the first item in the list, 'taskdefinition-vcp-netcore-webappdem01-build-79.release-9.json'. The list contains 17 items, each showing a file icon, the file name, and its corresponding release number.

| File  | Release                                 |
|---|---|
| taskdefinition-vcp-netcore-webappdem01-build-79.release-9.json  | new-task-definition-build-79.release-9  |
| taskdefinition-vcp-netcore-webappdem01-build-80.release-9.json  | new-task-definition-build-80.release-9  |
| taskdefinition-vcp-netcore-webappdem01-build-81.release-10.j... | new-task-definition-build-81.release-10 |
| taskdefinition-vcp-netcore-webappdem01-build-82.release-11.j... | new-task-definition-build-82.release-11 |
| taskdefinition-vcp-netcore-webappdem01-build-83.release-12.j... | new-task-definition-build-83.release-12 |
| taskdefinition-vcp-netcore-webappdem01-build-84.release-13.j... | new-task-definition-build-84.release-13 |
| taskdefinition-vcp-netcore-webappdem01-build-86.release-14.j... | new-task-definition-build-86.release-14 |
| taskdefinition-vcp-netcore-webappdem01-build-87.release-15.j... | new-task-definition-build-87.release-15 |
| taskdefinition-vcp-netcore-webappdem01-build-88.release-16.j... | new-task-definition-build-88.release-16 |
| taskdefinition-vcp-netcore-webappdem01-build-89.release-17.j... | new-task-definition-build-89.release-17 |
| ...   |   |
| taskdefinition-vcp-netcore-webappdem01-build-90.release-18.j... | new-task-definition-build-90.release-18 |
| taskdefinition-vcp-netcore-webappdem01-build-91.release-19.j... | new-task-definition-build-91.release-19 |
| taskdefinition-vcp-netcore-webappdem01-build-92.release-20.j... | new-task-definition-build-92.release-20 |
| taskdefinition-vcp-netcore-webappdem01-build-93.release-21.j... | new-task-definition-build-93.release-21 |
| taskdefinition-vcp-netcore-webappdem01-build-94.release-22.j... | new-task-definition-build-94.release-22 |
| taskdefinition-vcp-netcore-webappdem01-build-95.release-23.j... | new-task-definition-build-95.release-23 |
| taskdefinition-vcp-netcore-webappdem01-build-96.release-24.j... | new-task-definition-build-96.release-24 |
| taskdefinition-vcp-netcore-webappdem01-build-97.release-25.j... | new-task-definition-build-97.release-25 |
| taskdefinition-vcp-netcore-webappdem01-build-98.release-26.j... | new-task-definition-build-98.release-26 |
| taskdefinition-vcp-netcore-webappdem01-build-99.release-27.j... | new-task-definition-build-99.release-27 |

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**)

Update tasks: DEV
How deployment tasks work

What tasks need to happen to make this deployment a success

31 agents have the [capabilities](#) to deploy this environment

| Task Type   | Task Description  | How to use the Task                         |
|---|---|---|
| Clean working directory task  | Clean working directory   | How to use the Clean working directory task |
| Artifact download   | Download release contents   | How to use the Artifact download task       |
| Command   | Load release-image from artifacts   | How to use the Command task                 |
| Script  | Create directory for AWS resource definition repository   | How to use the Script task                  |
| Script  | Get AWS ECR Login info and export variables   | How to use the Script task                  |
| Docker  | Pull git-helper container-image from AWS ECR  | How to use the Docker task                  |
| Docker  | Clone aws-task-definitions repository thru git-helper docker tool                               | How to use the Docker task                  |
| Script  | Update AWS task definition  | How to use the Script task                  |
| Script  | Tag container image for AWS ECR   | How to use the Script task                  |
| Command   | Publish container to AWS ECR  | How to use the Command task                 |
| Command   | Set tag latest for image just pushed AWS ECR  | How to use the Command task                 |
| Docker  | Push new aws-task-definition to aws-resource-definitions repository thru git helper docker tool | How to use the Docker task                  |
| <b>Final tasks</b> Are always executed even if a previous task fails  |   |   |
| Command   | Remove release-image (Clean up)   | How to use the Command task                 |
| Command   | Remove git-helper docker tool container-image (Clean up)  | How to use the Command task                 |
| <a href="#" style="border: 1px solid #0070C0; padding: 2px 10px; color: inherit; text-decoration: none;">Add task</a> |   |   |

**Command configuration**

Task description

Remove git-helper docker tool container-image (Clean up)

Disable this task

Executable

docker

Argument

rmi 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform

Argument you want to pass to the command. Arguments with spaces in them must be quoted.

Environment variables

Extra environment variables. e.g. JAVA\_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space.

Working sub directory

Specify an alternative sub-directory as working directory for the task.

**Save**   [Cancel](#)

- **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=XXXXXXX"
$paramRepo = "&stashRepo=stash.mtv.com/scm/vcp/vcp-aws-resourcedefinitions.git"
$paramEmail = "&contactEmail=david.sorbona@vinn.com"
$paramAccount = "&RemoteAccountName=vinn-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.mtv.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.mtv.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>

COMEDY  
CENTRAL

**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.

**Settings**

**Repository permissions**

Repository permissions allow you to extend access to users and groups beyond that already granted via [project permissions](#).

**User access**

| Name          | Admin                               | Write                               | Read                                |
|---------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Add Users     |                                     |                                     | <input type="button" value="Read"/> |
| Hudson User   | <input type="checkbox"/>            | <input type="checkbox"/>            | <input checked="" type="checkbox"/> |
| David Sorbona | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |

### Procedure

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

**Environment: DEV**

**How you want to deploy**

Tasks define the steps involved in deploying a release to the environment.

**Edit tasks**

**Other environment settings**

These settings are not strictly necessary for your deployment to run, but they can help make Bamboo deployments go just right.

**Triggers 1** **Agents assignment** **Notifications 1** **Variables 10**

<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@vimm.com  | ×   |
| jenkinsJobName               | Docker-ECS-Update   | ×   |
| jenkinsServerUrl             | <a href="http://jenkins.us-east-1.aws.cloud.viacom.com:8080">http://jenkins.us-east-1.aws.cloud.viacom.com:8080</a> | ×   |
| 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)

|   |  |
|---|--|
| <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Clean working directory task</a> <br/> <span style="font-size: small;">Clean working directory</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Artifact download</a> <br/> <span style="font-size: small;">Download release contents</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Command</a> <br/> <span style="font-size: small;">Load release-image from artifacts</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Script</a> <br/> <span style="font-size: small;">Create directory for AWS resource definition repository</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Script</a> <br/> <span style="font-size: small;">Get AWS ECR Login info and export variables</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Docker</a> <br/> <span style="font-size: small;">Pull git-helper container-image from AWS ECR</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Docker</a> <br/> <span style="font-size: small;">Clone aws-task-definitions repository thru git-helper docker tool</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Script</a> <br/> <span style="font-size: small;">Update AWS task definition</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Script</a> <br/> <span style="font-size: small;">Tag container image for AWS ECR</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Command</a> <br/> <span style="font-size: small;">Publish container to AWS ECR</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Command</a> <br/> <span style="font-size: small;">Set tag latest for image just pushed AWS ECR</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Docker</a> <br/> <span style="font-size: small;">Push new aws-task-definition to aws-resource-definitions repository thru git-helper docker tool</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Docker</a> <br/> <span style="font-size: small;">Pull jenkins-helper container-image from AWS ECR</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Docker</a> <br/> <span style="font-size: small;">Send ECS Update job request to cloud orchestrator (Jenkins helper)</span> </div> <div style="background-color: #f0f0f0; padding: 5px; margin-bottom: 5px;"> <b>Final tasks</b> Are always executed even if a previous task fails </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Command</a> <br/> <span style="font-size: small;">Remove release-image (Clean up)</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <span style="color: #0070C0;">[ ]</span> <a href="#">Command</a> <br/> <span style="font-size: small;">Remove git-helper docker tool container-image (Clean up)</span> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px; background-color: #f0f0f0;"> <a href="#" style="color: inherit; text-decoration: none;">Add task</a> </div> | <p><b>Docker configuration</b></p> <p>Task description</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%;"> <span style="font-size: small;">Pull Jenkins-helper container-image from AWS ECR</span> </div> <p><input type="checkbox"/> Disable this task</p> <p>Command</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%;"> <span style="font-size: small;">Pull a Docker image from a Docker registry</span> </div> <p>The Docker command to execute</p> <p>Registry</p> <p><input type="radio"/> Docker Hub</p> <p><input checked="" type="radio"/> Custom registry</p> <p>Repository*</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%;"> <span style="font-size: small;">310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-toolbox-docker-jenkinshelper:latest</span> </div> <p>Registry address, repository name and optionally a tag to pull from the custom registry (e.g. 'registry.address:port/namespace/repository:tag')</p> <p><b>Credentials</b></p> <p>Leave these credentials empty to use the agent's <code>~/.dockercfg</code> credentials.</p> <p>Username</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%; height: 20px;"></div> <p>Password</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%; height: 20px;"></div> <p>Email</p> <div style="border: 1px solid #ccc; padding: 5px; width: 100%; height: 20px;"></div> <p><b>Advanced options</b></p> <hr/> <div style="display: flex; justify-content: space-between;"> <span style="color: #0070C0;">Save</span> <span>Cancel</span> </div> |
|---|--|

- **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)

The screenshot shows a Jenkins Pipeline configuration page. On the left, a sidebar lists various Jenkins tasks: Clean working directory task, Artifact download, Command, Script, Docker, Script, Docker, Script, Script, Command, Command, Docker, Docker, Docker, Final tasks, Command, Command. The "Send ECS Update job request to cloud orchestrator (Jenkins helper)" task is highlighted in blue. The main panel is titled "Docker configuration" and contains the following fields:

- Task description:** Send ECS Update job request to cloud orchestrator (Jenkins helper)
- Disable this task:**
- Command:** Run a Docker container
- Docker image:** 310827469077.dkr.ecr.us-east-1.amazonaws.com/viacomcontentplatform/vcp-
- Detach container:**
- Link to detached containers:**
- Container environment variables:** (empty field)
- Container Command:** ./request-jenkins-job\_check-status-until-ends.ps1 -JenkinsServerUrl "\${bamboo.jenkinsServerUrl}"
- Container working directory:** /scripts
- Additional arguments:** (empty field)
- Volumes:** (collapsible section)
- Advanced options:** (collapsible section)

At the bottom right are "Save" and "Cancel" buttons.

- 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)

The screenshot shows a Jenkins pipeline configuration. On the left, a vertical list of steps is displayed, including:

- Clean working directory task
- Artifact download
- Command
- Script
- Script
- Docker
- Docker
- Script
- Script
- Command
- Command
- Docker
- Docker
- Docker
- Final tasks
- Command
- Command
- Command

The last three steps are highlighted in blue, indicating they are part of the "Final tasks" section. The right side of the screen shows a detailed configuration panel for the selected "Command" step. The configuration includes:

- Command configuration**: A sub-section where the user can enter a command description.
- How to use the Command task**: A link to documentation.
- Task description**: A text input field containing "Remove jenkins-helper docker tool container-image (Clean up)".
- Disable this task**: A checkbox.
- Executable**: A dropdown menu set to "docker".
- Argument**: A text input field containing "amazonaws.com/viacomcontentplatform/vcp-toolbox-docker-jenkinshelper:latest".
- Environment variables**: An empty text input field.
- Extra environment variables**: A note stating "Extra environment variables. e.g. JAVA\_OPTS="-Xmx256m -Xms128m". You can add multiple parameters separated by a space."
- Working sub directory**: An empty text input field.
- Specify an alternative sub-directory as working directory for the task**: A note below the working directory field.

At the bottom of the configuration panel are "Save" and "Cancel" buttons.

- **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:

| Clean working directory task  |  |
|---|--|
| Clean working directory   |  |
| Artifact download   |  |
| Download release contents   |  |
| Command   |  |
| Load release-image from artifacts   |  |
| Script  |  |
| Create directory for AWS resource definition repository   |  |
| Script  |  |
| Get AWS ECR Login info and export variables   |  |
| Docker  |  |
| Pull git-helper container-image from AWS ECR  |  |
| Docker  |  |
| Clone aws-task-definitions repository thru git-helper docker tool                               |  |
| Script  |  |
| Update AWS task definition  |  |
| Script  |  |
| Tag container image for AWS ECR   |  |
| Command   |  |
| Publish container to AWS ECR  |  |
| Command   |  |
| Set tag latest for image just pushed AWS ECR  |  |
| Docker  |  |
| Push new aws-task-definition to aws-resource-definitions repository thru git-helper docker tool |  |
| Docker  |  |
| Pull jenkins-helper container-image from AWS ECR  |  |
| Docker  |  |
| Send ECS Update job request to cloud orchestrator (Jenkins helper)                              |  |
| Final tasks Are always executed even if a previous task fails                                   |  |
| Command   |  |
| Remove release-image (Clean up)   |  |
| Command   |  |
| Remove git-helper docker tool container-image (Clean up)  |  |
| Command   |  |
| Remove jenkins-helper docker tool container-image (Clean up)                                    |  |
| <a href="#">Add task</a>  |  |

### Clean working directory task configuration

Task description

Disable this task

[Save](#) [Cancel](#)

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.

### Update Service

A service lets you specify how many copies of your task definition to run. You could also use Elastic Load Balancing to distribute incoming traffic to the number of tasks running and coordinates task scheduling with the load balancer.

|                         |   |                   |
|-------------------------|---|-------------------|
| Task Definition         | vcp-netcore-webappdem01:33                        | <a href="#">i</a> |
| Cluster                 | CPE-Non-Production-ALB-Cluster-ECSCluster-17JM... | <a href="#">i</a> |
| Service name            | vcp-netcore-webappdem01                           | <a href="#">i</a> |
| Number of tasks         | 4   | <a href="#">i</a> |
| Minimum healthy percent | 25  | <a href="#">i</a> |
| Maximum percent         | 200   | <a href="#">i</a> |

## 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:

The screenshot shows the Jenkins Docker-ECS-Update dashboard. At the top, there's a navigation bar with links for 'Changes' and 'Recent Changes'. Below it is a 'Stage View' section with a timeline showing the progress of a build. The timeline includes stages like 'Get Approval' (13ms) and 'Deploy to ECS' (1min 4s). A specific build step is highlighted in green, labeled '#16735, Apr 09 19:11, No Changes'. To the left, there's a 'Build History' sidebar with a search bar and a list of recent builds: #16735 (Apr 9, 2017 6:11 PM), #16734 (Apr 9, 2017 6:10 PM), and #16733 (Apr 9, 2017 6:08 PM).

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      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      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      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      INFO: Jenkins job run status: SUCCESS
build 19-Apr-2017 06:13:53      SUCCESS: Jenkins job (Docker-ECS-Update / 17635) completed OK
build 19-Apr-2017 06:13:53
build 19-Apr-2017 06:13:53
FINISHED: Jenkins job run (Docker-ECS-Update / 17635) completed OK

```

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:

| Task                                 | Task Definition            | Group                           | Last Status | Desired Status |
|--------------------------------------|----------------------------|---------------------------------|-------------|----------------|
| 34db11bd-ddd5-4d95-ad16-c201e1b0ef71 | vcp-netcore-webappdem01:27 | service:vcp-netcore-webappdem01 | PENDING     | RUNNING        |
| b6b5be2c-8ea8-434b-b653-2adcc381a0cc | vcp-netcore-webappdem01:27 | service:vcp-netcore-webappdem01 | PENDING     | RUNNING        |
| aac265a4-4343-4da4-94c5-94d7142b0978 | vcp-netcore-webappdem01:25 | service:vcp-netcore-webappdem01 | RUNNING     | RUNNING        |
| b26dbe44-0992-472c-bfe1-c9943417b84  | vcp-netcore-webappdem01:25 | service:vcp-netcore-webappdem01 | RUNNING     | RUNNING        |

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

| Stage         | Average Stage Time |
|---------------|--------------------|
| Get Approval  | 3ms                |
| Deploy to ECS | 1min 52s           |

**Build History**

- #16736 (Apr 9, 2017 6:28 PM)
- #16735 (Apr 9, 2017 6:11 PM)
- #16734

## 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>

