
Table of Contents

Introduction	1.1
Operating Guidelines	1.2
Development Operating Procedures for Node.js GUI	1.3
Development Operating Procedures for Java Spring Boot or Node.js Microservices	1.4
Deploying a Node.js GUI	1.5
Deploying a Java Spring Boot or Node.js Microservice	1.6
Custom Configuration	1.7
Observability	1.8
Troubleshooting Guide	1.9
Local Development and Debugging	1.10
Pipeline Test Stages / Quality Gates	1.11
Code Examples	1.12
Platform Release Notes	1.13
Projected Release Notes	1.14
FAQ	1.15
Sample	1.16

Introduction

RefArch 1.5 Platform Operating Guidelines

- [RefArch 1.5 Platform Operating Guidelines](#)
 - [Dev Environments](#)
 - [Platform Release Schedule](#)
 - [Communication and Support](#)
 - [Upgrade Day](#)
 - [Post Upgrade Day](#)

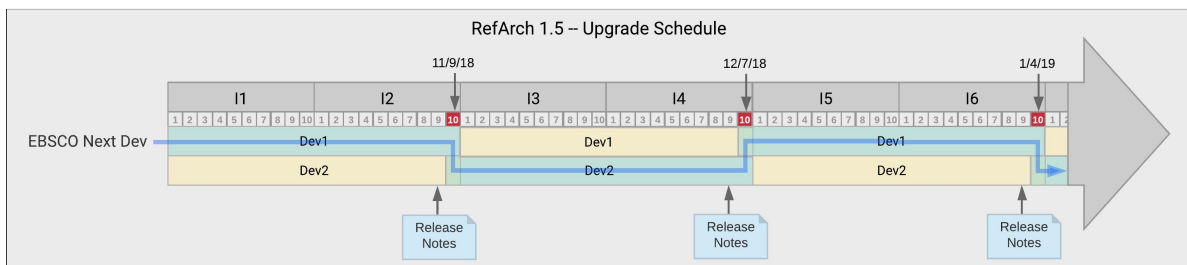
As a development team building services on the RefArch1.5 platform prior to GA release, you will be working on the platform as it is evolving. It will be important to follow the guidelines below to minimize disruptions and ensure we upgrade together on a cadence.

Dev Environments

- Microservices will be deployed and tested in 2 AWS dev environments to allow for platform improvements in one as active EBSCO Next platform development is going on in the other. In PI12, these will be the only environments available
 - **Current:** Dev teams deploy their microservices to this environment during normal iterations (green below)
 - **Next:** Dev teams may deploy their microservices to this environment in order to test changes required for the next version of the platform (yellow below)
- Current and Next environments will be rotated every other iteration in PI12. We will assess how this period works and adjust as necessary.
- Rotation will be executed with an automated rebuild, blowing away everything in both environments. We do this to ensure we can recreate environments in an automated way.
- Key non-breaking fixes and enhancements will be overlaid onto the Current environment as needed.

Platform Release Schedule

- Platform dev teams will release the Next environment by end of day Thursday, before Friday platform upgrade by product dev teams.



Communication and Support

Dev teams will use the resources below for instructions and to raise issues and questions about deployment to the Next version of the platform.

It is important we direct all discussion to the shared Teams channel so issues and solutions are available to all dev teams.

- Release Notes:

The [Platform Release Notes](#) section of this document will describe project changes required to upgrade to the new version of the platform. This will be complete by end of day Thursday, likely sooner.

- Teams room:

The [Ref Arch 1.5 Feedback](#) Teams Channel will include a Tab to capture status at various check-ins as well as use of the chat for ongoing issues reporting/progress

- Egress from microservices/mesh to external services is managed by submitting a Pull Request to a single repository (TBD). PR's will be reviewed and executed by "Platform Support."
- Ingress and routing to applications is done with Ambassador gateway (if ready) or by using host header (if not)

Upgrade Day

- Friday morning of upgrade day, product dev teams will
 - make any required changes to their code repos as identified in the [Platform Release Notes](#)
 - Perform a deployment
 - Deploying to the upgraded Next environment does not require a change to your Jenkinsfile -- starting the morning of Upgrade Day, all deployments will be targeted at the new environment.
 - There will be a way to override which environment (Dev1/Dev2) you deploy to but this should only be used in special cases in coordination with the Platform development teams.
- Check-ins with EA/PO will be held at 11am and 3pm and EA will be continuously monitoring the Teams channel to provide support
- EA will monitor teams room throughout the day and provide support
- In the event of an unforeseen problem with the Next environment, EA will make a call whether to fix forward (preferred) or roll back. EA will do their best to have a fix available by early afternoon for teams to react. Target end of day decision (3pm ET) with EA whether to roll back or fix forward.
- POs for teams (both platform dev and product dev) will monitor communication stream to coordinate & prioritize when teams need to address specific issues. (i.e. Set up conf call/MS Teams meeting for breakouts as needed for specific issues).
- Escalations during Upgrade Day should be routed to Aishwariya Bhavan; Seshu Pasam; Nate Baechtold via MS teams [Ref Arch 1.5 Feedback](#) channel
- Escalations during Iteration should be routed thru Respective Program team(s) (for prioritization) in addition to Aishwariya Bhavan; Seshu Pasam; Nate Baechtold
- Tracking of Issues:
 - Teams to use MS Teams channel to report immediate issues/progress with goal to resolve over the course of the day
 - Teams to use Rally team boards to add user stories for issues that teams expect to rollover into next iteration, to be considered in teams Planning meetings
- EPAM: ~2am ET start – end of day (off shore)
 - Krishna (EA in India)
 - Assist and monitor chat for EPAM to troubleshoot and collect info.

- Provide EPAM handoff at the end of the day to MS Team room, EPAM teams to ensure open communication on individual team rooms
- Local EA/PO to ensure communication back to EPAM teams end of day Friday for Monday morning update including awareness of solutions/remaining open issues
- Local: POs/Team members to monitor the public team room to understand issues, go/no go to start day.

Post Upgrade Day

- Cross ART team lessons learned in a consistent and formalized manner after each merge is key. EA to own scheduling the check in the first Monday in iteration 3, to include POs/EA/ADMs and tech leads as appropriate.

SDLC for Reference Architecture 1.5

- [Overview](#)
- [Introduction](#)
- [Logical SDLC Workflow](#)
- [Dev-Centric Culture](#)
- [Definitions](#)
- [GitHub Flow Branching Strategy](#)
- [Reference Branching Model](#)
- [SDLC Assertions](#)
- [Pipeline Quality Gates](#)
- [Reference Architecture 1.5 Pipelines](#)
- [Versioning](#)
- [Feature Toggles](#)
- [References](#)

Overview

This page defines the Reference Architecture 1.5 SDLC to be used by teams in the delivery of business value differentiating functionality via the Hydra-based pipelines. This is a living document, which will be continuously updated as information becomes available based on team experiences and input from technology, business, and PPMO organizations.

Introduction

This document defines the Software Development Life-cycle (SDLC) for the 1.5 Reference Architecture. The SDLC is the workflow a developer uses to deliver business value into Live environments. It defines the developer workflow, branching strategy, and best practices to be used when developing and releasing applications and services into our private and public cloud environments through the Hydra/Medusa based pipelines. The Reference Architecture 1.5 SDLC and branching strategy is informed by the AWS/Microservices Reference Architecture 1.0 and the On-Prem Reference Architecture.

AWS/Microservice Reference Architecture SDLC:

- [SDLC for Reference Architecture](#)
- [PLATFORM-207: Github Flow](#)

On-Prem Reference Architecture SDLC:

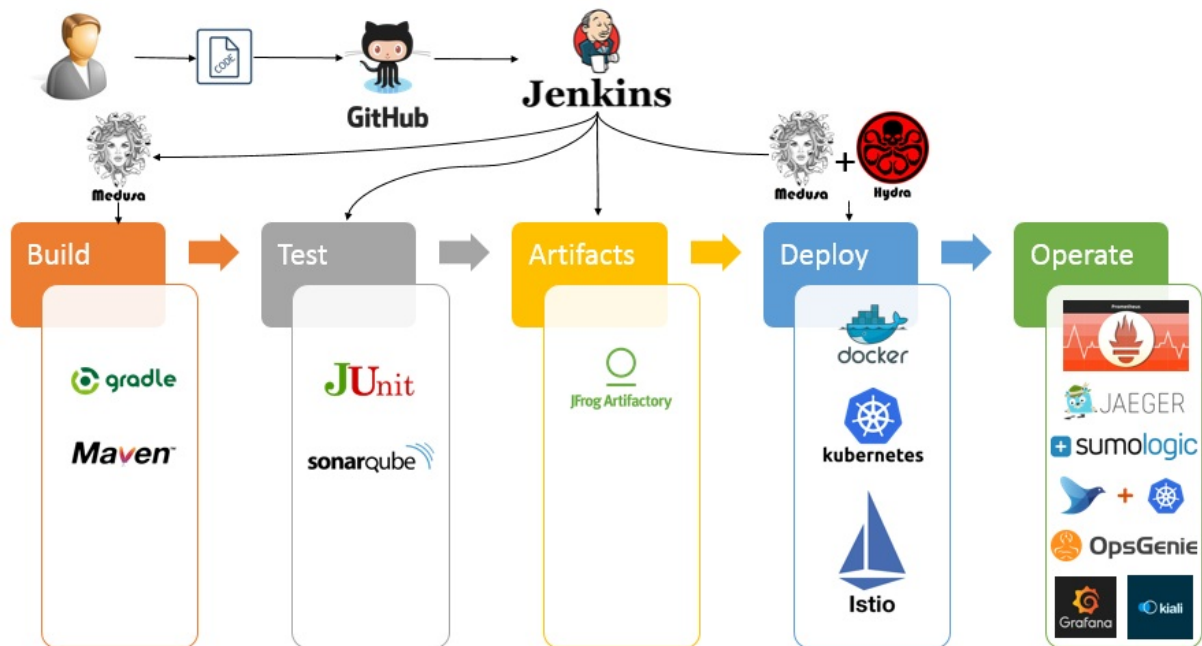
- [SDLC for On-Prem Reference Architecture](#)

Logical SDLC Workflow

A complex workflow blocks developer productivity. The following workflow, enforced through the use of the Medusa and Hydra pipeline libraries, creates a simple yet powerful workflow that enables:

- Faster delivery of business value into Live environments
- Enforcement of key NFRs, without any developer effort

- Complete consistency across all environments
- Dedicated Preview environment for acceptance testing prior to merge



Dev-Centric Culture

See: <http://confluence/display/entarch/Reference+Architecture#ReferenceArchitecture-DevCentricculture>

Extensive Collaboration and Shared Responsibility

- Partnership with Site Reliability Team, NOC, LiveOps in Monitoring Live
- Team owns ensuring Live viability for all changes promoted to Master

No Silos

- Co-Ownership with SRE, LiveOps of: Live monitoring of errors rates/dashboards, alerts, performance telemetry dashboards
- Quality is a concern of development (built-in quality)

Autonomous Teams

- Team owns creation of Pull Requests into Master.
- Team owns PR approvals and acts as Continuous Operations owner. In certain cases SA/Program Team level approval may be needed.

High Quality Development Process

- Team ensures all quality-first practices are followed
- CI processes provide fast feedback to teams
- CD processes enforce testing gates
- Teams leverage Preview environment for acceptance
- Teams build resiliency into systems

- circuit breakers
- zero downtime deployments

Valuing Feedback

- Teams look to measure everything and drive continuous improvement from metrics
- Teams value and respond quickly to defects reported from SRE, LiveOps and CustSat

Automation

- Teams build pipelines to Live following the Reference Architecture 1.5
- See: <https://github.com/EBSCOIS/platform.training.refarch1.5-devguide>

Definitions

- **Master** - EBSCOIS base repository. This repository is the starting point for all branches. All builds and deployments are sourced from this repository. Only the builds should have access to master. Only user story branches should be created here. Tags should be long-lived snapshots of the code.
- **User Story Branch Origin** - Personal repository. Each developer has their own origin. Developers can grant access to their origin as desired. Master branch should never contain local changes/work. User Story Branch Origins should always be kept in sync with the Master.
- **User Story Branch Local** - Local repository on the developers machine.
- **Developer Remotes** - Additional remotes added to collaboratively work with developers without triggering a build

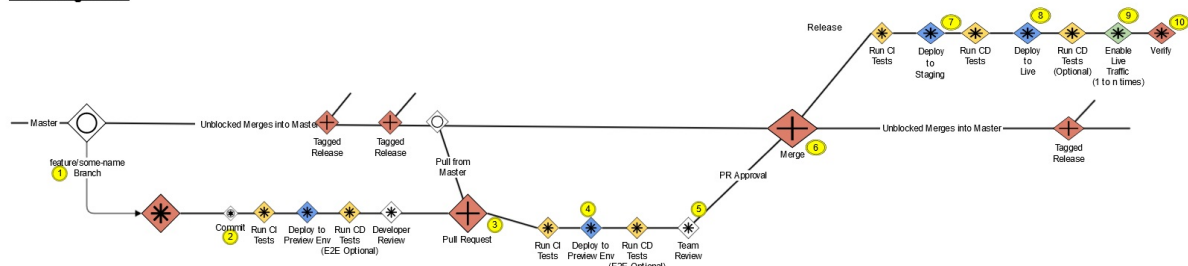
GitHub Flow Branching Strategy

The Branching Strategy for the Reference Architecture 1.5 follows the simplified GitHub Flow model that uses a "Branch and Merge" process:

<https://help.github.com/articles/github-flow/>

Reference Branching Model

Branching Model



#	Step	Description
1	Create feature/some-name Branch	Clone Master and create a tagged branch. Dev work scaled to two-week development and release.
2	Commit Changes	Make changes to local, run tests locally, then push to Branch Origin. Commits will trigger the deploy CI process triggers unit test run.

3	Create PR	The build system will automate the pull from master but merge conflicts will need to be resolved manually.
4	CI Pipeline publishes versioned artifact	Artifact pushed into Artifactory. CI tests (unit tests, integration tests, CDC) run before artifacts are published.
5	Preview Environment created	Used for feature/story acceptance testing and team demos. The Preview environment is automatically generated by creating a pull request. Preview environment should not be used for Performance tests. Preview environments are automatically cleaned up after a set time to live. A push can be used to recreate the Preview environment again if necessary.
6	Master Merge	PR approval required.
7	CD Pipeline Release into Staging Env	Follows Canary deployment pattern. Automated release verification performed. Automatic promotion if gate passes. Staging environment mirrors the Live environment and is suitable for running performance tests.
8	CD Pipeline Release into Live Env(s)	Follows Canary deployment pattern. Automated release verification performed.
9	Enable Live traffic	Automated promotion once all gates are passed. This step includes canary testing which is automated by Hydra and can happen from 1 to n times. This stage can also be used for dark traffic testing.
10	Review and Verify	Operations confirmed with live traffic. Validate Performance and Behavior.

SDLC Assertions

- All User Story Branches are created from Master
 - Do update (Pull down from Master) into your User Story branches frequently (At least daily)
 - Do merge to Master regularly (At least weekly)
- Master is kept in a releasable state at all times
 - The Master represents a collection of Live versions and intended releases
 - Do Not commit into Master directly
 - Do apply Hotfixes using a User Story Branch
 - Do Not rebase Master
- A Branch commit triggers a CI build
 - no versioned artifact created
 - unit tests are run
- A Pull Request triggers the creation of a Preview env
 - Quality gates must pass in this env
 - Team level Acceptance testing is done in this env
 - Do push user story branches for discussion prior to issuing a PR
 - PRs should represent small code changes that can be reviewed quickly
 - Do Not introduce breaking changes (follow SOLID, 12-factor)
- A Pull Request Approval triggers the merge into Master
 - PO/PM/SA acceptance drives PR approval
- A merge into Master triggers the CD process
 - Do Not merge in broken code
 - Do Not merge with conflicts (handle conflicts upon rebasing)
 - Do Tag all releases

Pipeline Quality Gates

Each delivery into an environment (Preview, Staging, Live) follows the Canary promotion methodology. All promotion events must be gated by the successful completion of a Quality Gate consisting of a set of tests appropriate for the environment. This helps to ensure that the version of the service being promoted into use within a given environment is fit-for-purpose.

Test Gates

For guidance on testing, including Consumer-Driven Contract testing and Performance testing, see:

https://github.com/EBSCOIS/platform.training.refarch1.5-devguide/blob/master/guides/Quality_Gates_Medusa.md

Stage Gates

Branch Builds

CI process initiated by a commit into a non-Master branch. Unit test gate is triggered.

Preview

Initiated by a commit in a feature branch or pull request. Preview environments last for a define set of time by default; please refer to [Hydra](#) document for time to live information. CI tests run. End-to-End tests are optional. Preview environment is constructed in the Staging cluster but in a cost-optimized way and should not be used for performance test. Components deployed to the Preview environment will work with other components already deployed in the Staging environment by default. However, the preview environment can also deploy additional components besides your own via an YAML configuration to provide an isolated environment to conduct testing and minimize interference to other teams.

Acceptance testing at the team level is done in this environment. PRs should not be approved until sufficient testing has been completed for the PO/PM/SA to accept the feature or user story.

Staging

CI process initiated by a merge to Master. The Staging environment is the next higher environment pipelines deliver into and mirrors the Live environment. It is the environment where full CD tests (E2E and Performance tests) can be executed.

Live

The Live environments are the final environments that the pipelines deliver into. The Live environments are where we need to ensure the newly deployed version is operationally viable. In this case it is acceptable to run a smaller set of "Smoke Tests" and CD tests to ensure that the newly deployed version operates as expected based on the major use cases supported by the component.

Versioning

[Semantic Versioning](#) should be used by all services being delivered into an environment.

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards-compatible manner, and
3. PATCH version when you make backwards-compatible bug fixes.

Feature Toggles

Note: Support for Feature Toggles in 1.5 will not be in place until after PI12.

Feature Toggles may be used to:

- isolate work in progress
- provide a mechanism to dynamically enable new business functionality post release
- enable canary testing of experimental algorithms
- enable A/B testing

Feature Toggle Best Practices:

- Feature Toggles must have a defined expiration date.
- Treat Feature Toggles as Technical Debt. Code supporting the feature toggle must be removed once the defined expiration date has been met.
- Testing must cover all Feature Toggle permutations - USE THEM SPARINGLY.

References

Reference Architecture Test Standards

- [Microservices Test Strategy](#)
- [SERVICES-260: Pipeline Testing](#)
- [UI-260: DevTest - Automated E2E testing](#)
- [SERVICES-270: Microservices Testing](#)
- [SERVICES-280: Testing Overview](#)

Sonarqube Quality Profiles:

<http://sonarqube73.eis-platformlive.cloud/profiles>

Reference Architecture 1.5 Feedback Channel

[Links to Teams Channel](#)

SDLC for Reference Architecture 1.5

Note: For the PI13.3 architecture drop, this document is specific to Java Spring Boot and Node.js edge microservices. It will remain so until the RefArch1.5 implementation is far enough along that the document becomes common to all RefArch1.5 supported technologies.

- [Overview](#)
- [Introduction](#)
- [Logical SDLC Workflow](#)
- [Dev-Centric Culture](#)
- [Definitions](#)
- [GitHub Flow Branching Strategy](#)
- [Reference Branching Model](#)
- [SDLC Assertions](#)
- [Pipeline Quality Gates](#)
- [Reference Architecture 1.5 Pipelines](#)
- [Versioning](#)
- [Feature Toggles](#)
- [References](#)

Overview

This page defines the Reference Architecture 1.5 SDLC to be used by teams in the delivery of business value differentiating functionality via the [Medusa](#)-based pipelines. This is a living document, which will be continuously updated as information becomes available based on team experiences and input from technology, business, and PPMO organizations.

Introduction

This document defines the Software Development Life-cycle (SDLC) for the 1.5 Reference Architecture. The SDLC is the workflow a developer uses to deliver business value into Live environments. It defines the developer workflow, branching strategy, and best practices to be used when developing and releasing applications and services into our private and public cloud environments through the [Medusa](#) based pipelines. The Reference Architecture 1.5 SDLC and branching strategy is informed by the AWS/Microservices Reference Architecture 1.0 and the On-Prem Reference Architecture.

AWS/Microservice Reference Architecture SDLC:

- [SDLC for Reference Architecture](#)
- [PLATFORM-207: Github Flow](#)

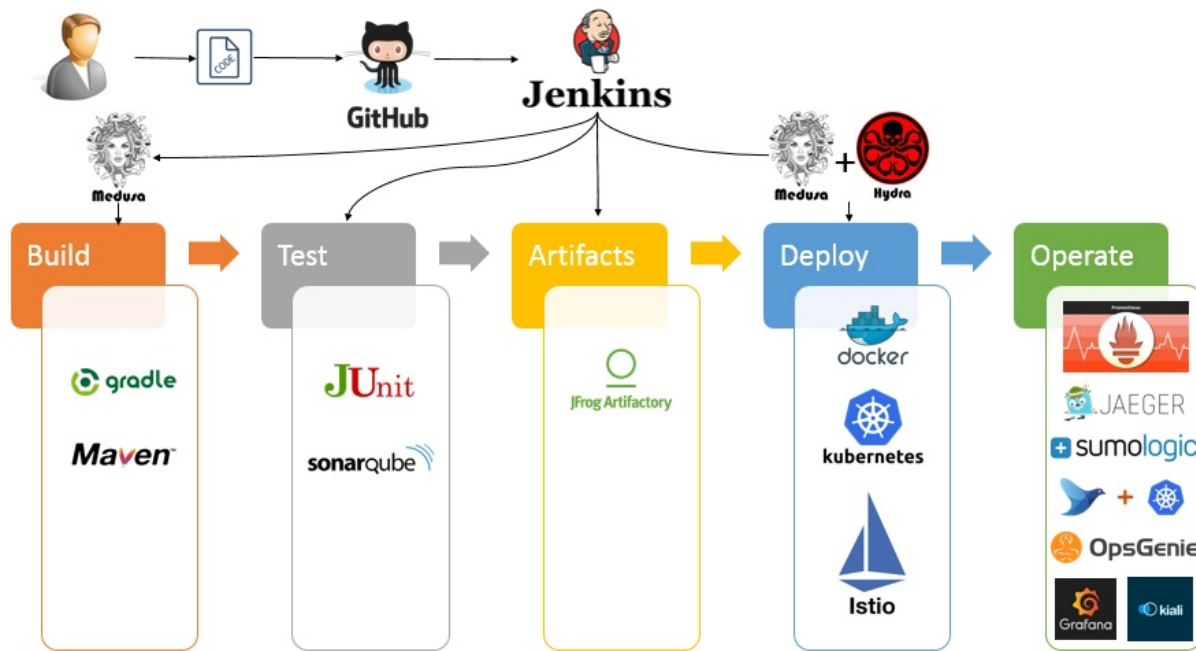
On-Prem Reference Architecture SDLC:

- [SDLC for On-Prem Reference Architecture](#)

Logical SDLC Workflow

A complex workflow blocks developer productivity. The following workflow, enforced through the use of the [Medusa](#) pipeline library, creates a simple yet powerful workflow that enables:

- Faster delivery of business value into Live environments
- Enforcement of key NFRs, without any developer effort
- Complete consistency across all environments
- Dedicated Preview environment for acceptance testing prior to merge



Dev-Centric Culture

See: <http://confluence/display/entarch/Reference+Architecture#ReferenceArchitecture-DevCentricculture>

Extensive Collaboration and Shared Responsibility

- Partnership with Site Reliability Team, NOC, LiveOps in Monitoring Live
- Team owns ensuring Live viability for all changes promoted to Master

No Silos

- Co-Ownership with SRE, LiveOps of: Live monitoring of errors rates/dashboards, alerts, performance telemetry dashboards
- Quality is a concern of development (built-in quality)

Autonomous Teams

- Team owns creation of Pull Requests into Master.
- Team owns PR approvals and acts as Continuous Operations owner. In certain cases SA/Program Team level approval may be needed.

High Quality Development Process

- Team ensures all quality-first practices are followed

- CI processes provide fast feedback to teams
- CD processes enforce testing gates
- Teams leverage Preview environment for acceptance
- Teams build resiliency into systems
 - circuit breakers
 - zero downtime deployments

Valuing Feedback

- Teams look to measure everything and drive continuous improvement from metrics
- Teams value and respond quickly to defects reported from SRE, LiveOps and CustSat

Automation

- Teams build pipelines to Live following the Reference Architecture 1.5
- See: <https://github.com/EBSCOIS/platform.training.refarch1.5-devguide>

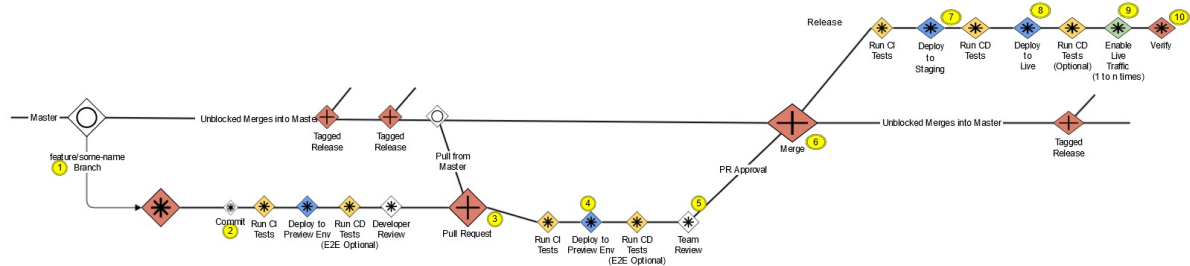
Definitions

- **Master** - EBSCOIS base repository. This repository is the starting point for all branches. All builds and deployments are sourced from this repository. Only the builds should have access to master. Only user story branches should be created here. Tags should be long-lived snapshots of the code.
- **User Story Branch Origin** - Personal repository. Each developer has their own origin. Developers can grant access to their origin as desired. Master branch should never contain local changes/work. User Story Branch Origins should always be kept in sync with the Master.
- **User Story Branch Local** - Local repository on the developers machine.
- **Developer Remotes** - Additional remotes added to collaboratively work with developers without triggering a build

GitHub Flow Branching Strategy

- The Branching Strategy for microservices in the Reference Architecture 1.5 follows a variation of the simplified GitHub Flow model that uses a "Branch and Merge" process: <https://help.github.com/articles/github-flow/>
 - **Note:** the variation from the simplified GitHub Flow model is that we do not make use of `release/` branches. We release directly from the master branch.
 - The format of release branch names is very specific. It is described in the [Repository Branches section of the Medusa readme](#).
- Infrastructure projects do use `Release` branches in order to support the "drop" cadence model we are using for early platform development.
- **Note** all branches must be named according to the rules described in the [Repository Branches section of the Medusa readme](#)

Reference Branching Model

Branching Model

#	Step	Description
1	Create User Story Branch	Clone Master and create a tagged branch. Dev work scaled to two-week development and release.
2	Commit Changes	Make changes to local, run tests locally, then push to Branch Origin. CI process triggers test run.
3	CI Pipeline publishes versioned pre-release artifact	Artifact pushed to ECS.
4	Preview Environment created	Used for developer testing.
5	Create PR	
6	CI Pipeline publishes versioned pre-release artifact	Artifact pushed to ECS.
7	Preview Environment created	Used for feature/story acceptance testing and team demos.
8	Master Merge	PR approval required.
9	CI Pipeline publishes versioned release artifact	Artifact pushed to ECS.
10	CD Pipeline Release into Integration Env	Follows Canary deployment pattern. Automated release verification performed. Automatic promotion if gate passes.
12	CD Pipeline Release into Live Env(s)	Follows Canary deployment pattern. Automated release verification performed.
13	Enable Live traffic	Automated promotion once all gates are passed.
14	Review and Verify	Operations confirmed with live traffic. Validate Performance and Behavior.

SDLC Assertions

- All User Story Branches are created from Master
 - Do update (Pull down from Master) into your User Story branches frequently (At least daily)
 - Do merge to Master regularly (At least weekly)
- Master is kept in a releasable state at all times
 - The Master represents a collection of Live versions and intended releases
 - Do Not commit into Master directly
 - Do apply Hotfixes using a User Story Branch
 - Do Not rebase Master
- A push to a public branch triggers a CI build
 - A versioned pre-release artifact is created
 - Unit tests are run (as determined by archetype)
 - Integration tests are run (as determined by archetype)

- A push to a public branch triggers the creation of a Preview env
 - Quality gates must pass in this env
 - Preview environments live for only 2 hours
- A Pull Request triggers the creation of a Preview env
 - Quality gates must pass in this env
 - Static analysis is run (as determined by archetype)
 - Team level Acceptance testing is done in this env
 - Do push user story branches for discussion prior to issuing a PR
 - PRs should represent small code changes that can be reviewed quickly
 - Do Not introduce breaking changes (follow SOLID, 12-factor)
- A Pull Request Approval triggers the merge into Master
 - PO/PM/SA acceptance drives PR approval
- A merge into Master triggers the CD process
 - Do Not merge in broken code
 - Do Not merge with conflicts (handle conflicts upon rebasing)
 - All releases are tagged automatically by pipeline

Pipeline Quality Gates

Each delivery into an environment (Preview, Int, Live) follows the Canary promotion methodology supported by [Hydra](#). All promotion events must be gated by the successful completion of a Quality Gate consisting of a set of tests appropriate for the environment. This helps to ensure that the version of the service being promoted into use within a given environment is fit-for-purpose.

Test Gates

For guidance on testing, including Consumer-Driven Contract testing and Performance testing, see: [Microservices Test Strategy](#)

Sonarqube Static Code Analysis

Gate consists of static code analysis using Sonarqube. Outcome of this stage is a report that teams will use to assess code quality to help drive identification of tech debt stories for backlog. Standard Sonarqube Quality Profiles have been established for a number of different programming languages. See this Yammer post: [Yammer](#)

Unit Test Gate

All unit tests associated with the project are run. Any test failure blocks pipeline progress.

Integration Test Gate

All integration and/or contract tests associated with the project are run. Any test failure blocks pipeline progress.

End-to-End Test Gate

All long running end-to-end tests associated with the project are run. Any test failure blocks pipeline progress.

Performance Test Gate

Performance tests are run by mirroring Live traffic. Gate compares metrics from new build against established baselines. Gate fails if performance is not within a specified margin. Failure blocks Canary promotion in INT and Live environments.

Stage Gates

PR/feature Branch Builds

CI process initiated by a commit into a PR or feature branch. The following gates are triggered and a preview environment is created:

- Unit tests.
- Contract tests.
- Integration tests.
- Sonarqube Static Code Analysis.
- End-to-End tests (against service Preview)

Master Branch Builds

CI process initiated by a PR merge to Master. The following gates are triggered:

- Unit tests.
- Contract tests.
- Integration tests.
- Sonarqube Static Code Analysis.
- End-to-End tests.
- Performance tests.

Preview

Initiated by the creation of a PR.

Acceptance testing at the team level is done in this environment. PRs should not be approved until sufficient testing has been completed for the PO/PM/SA to accept the feature or user story.

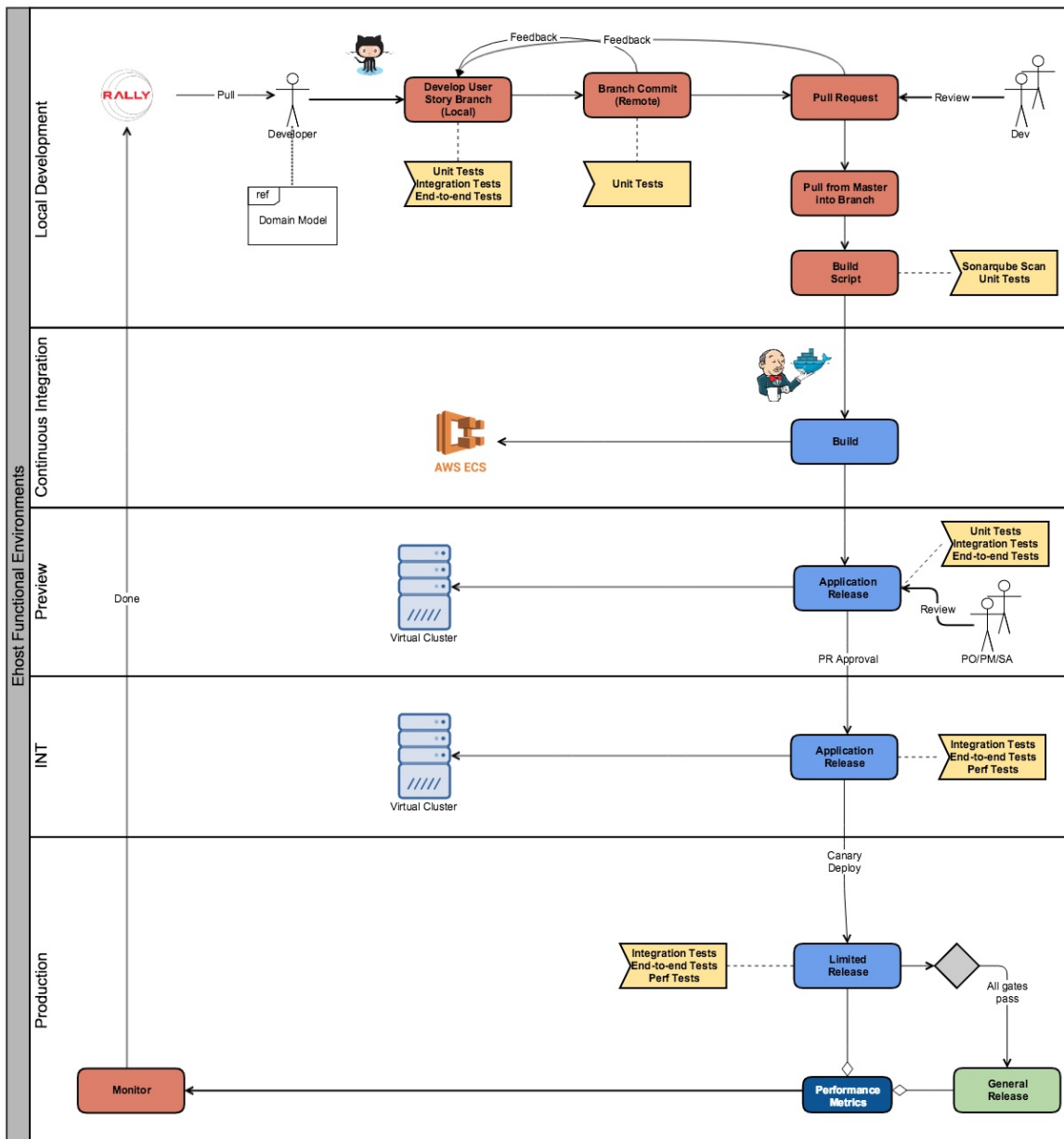
INT

The INT environment is the first "real" Assembly Line environment pipelines deliver into. End-to-end integration and performance tests are executed.

Live

The Live environments are the final environments that the pipelines deliver into. The Live environments are where we need to ensure the newly deployed version is operationally viable. End-to-end integration and performance tests are executed.

Reference Architecture 1.5 Pipelines



Notes:

- Developers should run tests locally before committing changes to a remote branch.
- A check-in to a remote branch triggers a CI build. Unit & Integration tests are run (as determined by the archetype).
- A PR into Master triggers the generation of a versioned pre-release artifact (that represents a merging the branch and master), and deployment of that artifact in a Preview environment.
- Team/ART level Acceptance testing is performed in the PR Preview environment.
- PR approval and merge triggers a build of the master branch resulting in deployment into INT and Live.
- Canary deployment used in all environments:
 - All automated gates must pass before full promotion

Versioning

[Semantic Versioning](#) is used by all services being delivered into an environment.

Given a version number MAJOR.MINOR.PATCH, versioning is handled in the following manner:

1. MAJOR version must be incremented when you make incompatible API changes.
 - Developers have control of this part of the version. It can be set in the [applicationDefinition.yaml](#) file.
 - Strictly speaking however, there should be no need to change this part of the version when developing a microservice, because any breaking change should be deployed as a new service out of a new project repo, e.g. breaking changes in a service named `myService` should be made in a new project repo resulting in `myService2`.
2. MINOR version increments automatically as part of the build process. Developers have no control over this part of the version number. For the purposes of pipeline automation no distinction is made between "new features" and "bug fixes" with respect to the MINOR version number.
3. PATCH version does not change. It serves no specific purpose other to remain compatible with systems and services that require a 3 part version.

Feature Toggles

Note: Support for Feature Toggles in 1.5 will not be in place until after PI13.

[Feature Toggles](#) may be used to:

- isolate work in progress
- provide a mechanism to dynamically enable new business functionality post release
- enable canary testing of experimental algorithms
- enable A/B testing

Feature Toggle Best Practices:

- Feature Toggles must have a defined expiration date.
- Treat Feature Toggles as Technical Debt. Code supporting the feature toggle must be removed once the defined expiration date has been met.
- Testing must cover all Feature Toggle permutations - USE THEM SPARINGLY.

References

Reference Architecture Test Standards

- [Microservices Test Strategy](#)
- [SERVICES-260: Pipeline Testing](#)
- [UI-260: DevTest - Automated E2E testing](#)
- [SERVICES-270: Microservices Testing](#)
- [SERVICES-280: Testing Overview](#)

Sonarqube Quality Profiles:

<http://sonarqube73.eis-platformlive.cloud/profiles>

Reference Architecture 1.5 Feedback Channel

[Link to Teams Channel](#)

Deploying a service in RefArch1.5

Deploying a service in RefArch1.5

- Environments
 - DEV Clusters Info
 - DEV5
 - Jenkinsfile updates to target the dev environment, for teams NOT USING MEDUSA
 - Access
 - SANDBOX Cluster Info
 - Links
 - Access
- Using Jenkins Pipeline and Hydra (to deploy to dev environment)
 - Deployment Manifest [hydra/default.yaml]
 - Application Definition File [applicationdefinition.yaml]
 - Jenkinsfile
- What happens during deployment
- Using Hydra CLI (ONLY use for sandbox environment)
- Verifying & Accessing Deployed Service
- Generating traffic to deployed services
 - Role of synthetic/live traffic for hydra deployments
 - Automated traffic generation in the cluster
- Troubleshooting

Environments

DEV Clusters Info

DEV5

- Name: **EksJ-VpcA-useast1-DeliveryDevQA-Dev5**
- AWS account: eis-deliverydevqa
- AWS region: us-east-1
- [~/.kube/config](#) file
- Links:
 - Kiali: <http://kiali.eksj-useast1.eks.eis-deliverydevqa.cloud:20001/kiali>
 - Index page with link to other observability tools: <http://monitoring.eksj-useast1.eks.eis-deliverydevqa.cloud>

Jenkinsfile updates to target the dev environment, for teams NOT USING MEDUSA

- Use the **@refarch1.5-current** branch of the platform.infrastructure.pipelinelibrary
 - e.g., in jenkinsfile the second line would be:
 - **@Library("platform.infrastructure.pipelinelibrary@refarch1.5-current") _**

Access

- ADFS-ExDevTeamRole gives read-only access to the cluster. This would be the default for all teams/devs.
- Additionally, ADFS-ExAdministratorsRole gives admin access to the cluster. This is for EA/admins to manage the

cluster.

SANDBOX Cluster Info

- Name: **Eksl-VpcA-useast1-DeliveryDevQA-EASandbox** -- Note: this is the **SANDBOX** environment for PI12
- AWS account: eis-deliverydevqa
- AWS region: us-east-1
- [~/kube/config file](#)

Links

- Kiali: <http://kiali.eksi-useast1.eks.eis-deliverydevqa.cloud:20001> (username/pwd: admin/admin)
- Index page with link to other observability tools: <http://monitoring.eksi-useast1.eks.eis-deliverydevqa.cloud>

Access

- ADFS-ExDevTeamRole gives admin access to the cluster. **Please exercise caution when executing commands that could potentially delete resources on this cluster.**

Using Jenkins Pipeline and Hydra (to deploy to dev environment)

Deployment in RefArch 1.5 is done using a Jenkins Pipeline specified in a Jenkinsfile in the root of your Git repository. Rather than having dev teams write their own Jenkinsfile (as in RefArch 1.0), a common one will be provided for inclusion into code repositories. This will contain all the necessary stages for build, test, quality gates, canary deployments, etc. and will allow for extension where necessary.

For early access, before the common Jenkinsfiles are available for Java/Spring and Javascript/Node.js, you can use a preliminary pipeline (like the one specified below) to orchestrate deployment of microservices to RefArch1.5 DevQA infrastructure i.e., Kubernetes + Istio.

Pipelines in RefArch 1.5 use pipeline functions from the [Hydra Project](#) to execute deployments. Hydra relies on several files in your repo which specify how and where your project should be deployed. Below are the dependencies a development team deploying services on the RefArch1.5 platform will need to address to use the hydra functions in their pipeline:

Deployment Manifest [hydra/default.yaml]

- This is the default deployment manifest file used by hydra, if an {env-name}.yaml file is not found.
- For more information on supported stages and customization of this file, please refer to: [Hydra Deployment Manifest](#)

Application Definition File [applicationdefinition.yaml]

- This file is required for tagging resources for cost appropriation and cloudability reporting.
- The file should have the following key-value pairs at minimum: Microservice:

```
Market      : training
Domain      : platform
Application : bookprovider
Team        : paws.ea
```

For valid values for these tags, please refer to: [Application Metadata Definition](#)

Jenkinsfile

- Add the following stages to the Jenkinsfile to deploy to refarch 1.5 (replacing the current deploy stages)

hydraPreview()

- this step applies to feature branches only, not executed on master.
- the feature branches will need to have an open PR for preview to work.
- the link to the preview environment is posted as a comment on the open PR in github.

hydraDeploy deployEnv: "dev"

- this step applies ONLY to the master branch, not executed on feature branches.
- the deployEnv argument determines the target environment where the service is deployed. Hydra looksups the environment-to-eks cluster mapping using this file: [cluster_mapping.yaml](#)
- See this sample Jenkinsfile to get started: [Jenkinsfile for a simple SpringBoot service](#)

What happens during deployment

The following resources are created for the microservice, by Hydra as part of the hydraDeploy command:

Kubernetes resources/controllers

- [Deployment](#)
- [ReplicaSet](#)
- [horizontal-pod-autoscaler](#) aka. [hpa](#)
- [ConfigMap](#)

Istio resources

- [VirtualService](#)
- [DestinationRule](#)

Using Hydra CLI (ONLY use for sandbox environment)

Projects deployed into DevQA use a pipeline as above to automate deployment. For the sandbox environment, you can use the Hydra CLI from the command line to perform manual deployments:

- Get pipenv on your workstation, if you do not have it already [macOS] `sudo pip install pipenv`
- Install hydra as a local package
 - [hydra] `pipenv install -e .`
- Run hydra commands
 - [hydra] `pipenv run hydra`
 - usage: `hydra {deploy,preview,teardown,reset,alert} ...`
 - example:
 - `hydra deploy [-h] [-m MANIFEST] -n NAME -ns NAMESPACE -i IMAGE -v`

```
VERSION -e ENV -t TEAM [-c CLUSTER] [-l KEY=VALUE]
```

Verifying & Accessing Deployed Service

A `VirtualService` is automatically created for the deployed application, to allow routing requests to the application pods/containers. The hosts that are allowed to access the service by default are of the format `service-name.external-dns-hosted-zone-for-cluster`.

For example, if the service 'simplemiddle' is deployed to the cluster with dns entry eksj-useast1.eks.eis-deliverydevqa.cloud, you can access the endpoint /simplemiddle/search via curl as follows: curl -HHost:simplemiddle.eksj-useast1.eks.eis-deliverydevqa.cloud <http://istio.eksj-useast1.eks.eis-deliverydevqa.cloud/simplemiddle/search?q=moon>

Note that you are using the -H host header with the request - this is to allow the test client to impersonate the DNS binding for that host and access the service.

Generating traffic to deployed services

Role of synthetic/live traffic for hydra deployments

- If a previous version of the service is already present in the cluster, hydra deploy will automatically attempt the canary stage
- The metrics generated from any real/live traffic are used to validate the health of the deployment
- Here is the default configuration for the canary stage in hydra's deployment manifest:

```
Stages:
- Type: Canary
  Duration: 1m
  Weight: 30

- Type: Canary
  Duration: 1m
  Weight: 70

Validation:
  MinimumSamples: 50
  Metrics:
    - Type: Error
      PercentErrors: 5
      PercentIncrease: 1

    - Type: Performance
      Duration: 1000
      PercentIncrease: 5
```

- To summarize:
 - No synthetic/real traffic is needed for the **first deployment** of a service
 - **2nd and all subsequent deployments** require synthetic/real traffic to generate metrics for the validations at the canary stage

Automated traffic generation in the cluster

- To address this requirement for metrics-driven deployments and automated promotion, as well as a means to proactively run passive health checks for deployed services, we have a `loadgens3` application running in the cluster.

NOTE: This is a *beta* implementation to assist with traffic generation in the cluster and the workflow will likely be further refined as the 1.5 platform evolves.

- How does `loadgens3` work?
 - This is a simple python application, which has been dockerized to run in the kubernetes cluster
 - The application's repo can be found here: [platform.infrastructure.ekscluster-loadgen-s3](https://github.com/platform-infrastructure/ekscluster-loadgen-s3)
 - `loadgens3` loops through a list of urls from an `s3 bucket` every 5 seconds, to generate light traffic against

deployed services in the cluster.

- An API exists via API Gateway, that calls lambda functions to update the S3 bucket.
- How do teams use `loadgens3` app to help generate traffic for their service?
 - To add an endpoint for your service in order to generate steady, light traffic against it in the cluster, you can do this via the loadgen UI
`http://loadgen-fe.eksj-useast1.eks.eis-deliverydevqa.cloud`
 - The lists of services that it is generating load for are split by cluster.
 - To add a service, select the + from the left menu. Enter the name of your service, the URL that you want traffic directed to and select the cluster from the dropdown. Press Save Configuration to save.
 - You should then see your services in the main page.
 - If you need to delete, find your service in the list, and press the delete button.
 - If you need to amend a service, delete it, then recreate it.
 - Please see the [loadgens3 repo#readme](#) for more information.
 - To monitor the activity against your service, you can utilize the various [Observability](#) dashboards/tools, as well as review the logs on the `loadgens3` container using this command: `k logs -f --tail=100 deployment/loadgens3`
 - **NOTE:** This action will only be required *once* for a service, unless the endpoint changes. Once your service endpoint has been added i.e., after the 1st deployment, subsequent deployments will be able to use the metrics generated out of this activity for canary deployments and no further action is needed for traffic generation.

Troubleshooting

- If the hydra deploy command fails (cli or jenkins-based) due to an unhandled exception, there may be orphaned deployments/other k8s and Istio resources in the cluster that would need to be cleaned up manually. Please contact EA to help reset these.
- Please contact EA on the [RefArch1.5 Chat Set - Troubleshooting Channel](#) if this happens to help diagnose the failure scenario and identify next steps. You can @mention [Aish](#), [Nate](#) & [Seshu](#) to get someone to look into it quickly.

Deploying a service in RefArch1.5

Note: For the PI13.I3 architecture drop, this document is specific to Java Spring Boot and Node.js edge microservices. It will remain so until the RefArch1.5 implementation is far enough along that the document becomes common to all RefArch1.5 supported technologies.

Deploying a service in RefArch1.5

- [Environments](#)
 - [DEV Clusters Info](#)
 - [DEV5](#)
 - [Jenkinsfile updates to target the dev environment](#)
 - [Access](#)
 - [SANDBOX Cluster Info](#)
 - [Links](#)
 - [Access](#)
- [Using Jenkins Pipeline and Hydra \(to deploy to dev environment\)](#)

```
- [Deployment Manifest [ `pipeline/deploymentmanifests/manifest.yaml` ]](#deployment-manifest-pipelinedeplo
ymentmanifestsmanifestyaml)
- [Application Definition File [ `applicationDefinition.yaml` ]](#application-definition-file-applicationde
finitionyaml)
- [Stub Services Manifest [ `pipeline/stubRunner/servicesManifest.yaml` ]](#stub-services-manifest-pipeline
stubrunnerservicesmanifestyaml)
```

- [What happens during deployment](#)
- [Using Hydra CLI \(ONLY use for sandbox environment\)](#)
- [Verifying & Accessing Deployed Service](#)
- [Generating traffic to deployed services](#)
 - [Role of synthetic/live traffic for hydra deployments](#)
 - [Automated traffic generation in the cluster](#)
- [Troubleshooting](#)

Environments

DEV Clusters Info

DEV5

- Name: **EksJ-VpcA-useast1-DeliveryDevQA-Dev5** -- Note: this is the **DEV5** environment for PI13
- AWS account: eis-deliverydevqa
- AWS region: us-east-1
- [~/kube/config file](#)
- Links:
 - Kiali: <http://kiali.eksj-useast1.eks.eis-deliverydevqa.cloud.:20001/kiali> (username/pwd: admin/admin)
 - Index page with link to other observability tools: <http://monitoring.eksj-useast1.eks.eis-deliverydevqa.cloud>

Jenkinsfile updates to target the dev environment

- All teams deploying to dev clusters in pi13 must use [medusa:snake:](#).
- To setup a Medusa pipeline, teams must adhere to the instructions in the [medusa:snake: README.md](#).

Access

- ADFS-ExReadOnlyRole gives read-only access to the cluster. This would be the default for all teams/devs.
- Additionally, ADFS-ExAdministratorsRole gives admin access to the cluster. This is for EA/admins to manage the cluster.

SANDBOX Cluster Info

- Name: **EksI-VpcA-useast1-DeliveryDevQA-EASandbox** -- Note: this is the **SANDBOX** environment for PI13
- AWS account: eis-deliverydevqa
- AWS region: us-east-1
- [~/kube/config file](#)

Links

- Kiali: <http://kiali.eksi-useast1.eks.eis-deliverydevqa.cloud:20001> (username/pwd: admin/admin)
- Index page with link to other observability tools: <http://monitoring.eksi-useast1.eks.eis-deliverydevqa.cloud>

Access

- ADFS-ExDevTeamRole gives admin access to the cluster. **Please exercise caution when executing commands that could potentially delete resources on this cluster.**

Using Jenkins Pipeline and Hydra (to deploy to dev environment)

Deployment in RefArch 1.5 is done using a Jenkins Pipeline. The [supported Jenkinsfile is described in detail in the medusa:snake: README.md](#)

Medusa pipelines in RefArch 1.5 use pipeline functions from the [Hydra Project](#) to execute deployments. Medusa and Hydra rely on several files in your repo which specify how and where your project should be deployed. Below are the dependencies a development team deploying services on the RefArch1.5 platform will need to address:

Deployment Manifest [`pipeline/deploymentmanifests/manifest.yaml`]

- This is the default deployment manifest file used by hydra, if an {env-name}.yaml file is not found.
- For more information on supported stages and customization of this file, please refer to: [Hydra Deployment Manifest](#)

Application Definition File [`applicationDefinition.yaml`]

- This file is required for tagging resources for cost appropriation and Cloudability reporting.
- It also provides certain pipeline specific input data.
- The file should conform to the [application definition description in the medusa:snake: README.md](#)

Stub Services Manifest [`pipeline/stubRunner/servicesManifest.yaml`]

- **NOT applicable to Java Spring Boot services, which manage stubs natively through Gradle**
- This is a simple key:value manifest, which lists your runtime service dependencies and their Spring Cloud Contract stub artifacts (generated from Contract tests in the provider project)
- VALUE is the stub name in Artifactory

- KEY is an arbitrary name, consisting of upper-case letters, digits and underscores, for referring to the service dependency in your application code. During integration/contract testing, Medusa will spin up each stub and provide its location to your application via an environment variable in the form of `[KEY]_ORIGIN`.
- Please refer to the [node-express-cookiecutter](#) project for an example of [the manifest](#) and how one might [utilize the env var in application code](#)

What happens during deployment

The following resources are created, for the microservice, by Hydra as part of the hydra prepare and deploy commands:

Kubernetes resources/controllers

- [Deployment](#)
- [ReplicaSet](#)
- [horizontal-pod-autoscaler](#) aka. [hpa](#)
- [ConfigMap](#)

Istio resources

- [VirtualService](#)
- [DestinationRule](#)

Using Hydra CLI (ONLY use for sandbox environment)

Projects deployed into DevQA use a pipeline as described above to automate deployment.

Verifying & Accessing Deployed Service

A `VirtualService` is automatically created for the deployed application, to allow routing requests to the application pods/containers. The hosts that are allowed to access the service by default are of the format *service-name.external-dns-hosted-zone-for-cluster*.

For example, if the service 'simplemiddle' is deployed to the cluster with dns entry `eksj-useast1.eks.eis-deliverydevqa.cloud`, you can access the endpoint `/simplemiddle/search` via curl as follows: `curl -HHost:simplemiddle.eksj-useast1.eks.eis-deliverydevqa.cloud http://istio.eksj-useast1.eks.eis-deliverydevqa.cloud/simplemiddle/search?q=moon`

Note that you are using the `-H` host header with the request - this is to allow the test client to impersonate the DNS binding for that host and access the service.

Generating traffic to deployed services

Role of synthetic/live traffic for hydra deployments

- If a previous version of the service is already present in the cluster, hydra deploy will automatically attempt the canary stage
- The metrics generated from any real/live traffic are used to validate the health of the deployment
- Here is the default configuration for the canary stage in hydra's deployment manifest:

```
Stages:
```

```

- Type: Canary
  Duration: 1m
  Weight: 30

- Type: Canary
  Duration: 1m
  Weight: 70

Validation:
  MinimumSamples: 50
  Metrics:
    - Type: Error
      PercentErrors: 5
      PercentIncrease: 1

    - Type: Performance
      Duration: 1000
      PercentIncrease: 5

```

- To summarize:
 - No synthetic/real traffic is needed for the **first deployment** of a service
 - **2nd and all subsequent deployments** require synthetic/real traffic to generate metrics for the validations at the canary stage

Automated traffic generation in the cluster

- To address this requirement for metrics-driven deployments and automated promotion, as well as a means to proactively run passive health checks for deployed services, we have a `loadgens3` application running in the cluster.

NOTE: This is a *beta* implementation to assist with traffic generation in the cluster and the workflow will likely be further refined as the 1.5 platform evolves.

- How does `loadgens3` work?
 - This is a simple python application, which has been dockerized to run in the kubernetes cluster
 - The application's repo can be found here: [platform.infrastructure.ekscluster-loadgen-s3](https://github.com/platform-infrastructure/ekscluster-loadgen-s3)
 - `loadgens3` loops through a list of urls from an `s3 bucket` every 5 seconds, to generate light traffic against deployed services in the cluster.
 - An API exists via API Gateway, that calls lambda functions to update the S3 bucket.
- How do teams use `loadgens3` app to help generate traffic for their service?
 - To add an endpoint for your service in order to generate steady, light traffic against it in the cluster, you have 2 options:

1. You can use the `PATCH` operation against the `/url` path in this swagger-ui endpoint <http://loadgen.eis-deliverydevqa.cloud/>, with the following payload:

```

{
  "service-name": "string",
  "service-url": "string"
}

```

2. You can make a `CURL` command directly to the API gateway endpoint like this:

```

curl -X PATCH \
  https://wfasypdjb.execute-api.us-east-1.amazonaws.com/beta/url \
  -H 'Cache-Control: no-cache' \
  -H 'Content-Type: application/json' \
  -d '{"service-url" : "http://istio.eksj-useast1.eks.eis-deliverydevqa.cloud", "service-name" : "EB-SCONEXT-UI"}'

```

- Please see the [loadgens3 repo#readme](#) for more information.
- To monitor the activity against your service, you can utilize the various [Observability](#) dashboards/tools, as well as review the logs on the `loadgens3` container using this command: `k logs -f --tail=100 deployment/loadgens3`
- **NOTE:** This action will only be required *once* for a service, unless the endpoint changes. Once your service endpoint has been added i.e., after the 1st deployment, subsequent deployments will be able to use the metrics generated out of this activity for canary deployments and no further action is needed for traffic generation.

Troubleshooting

- If the hydra deploy command fails (cli or jenkins-based) due to an unhandled exception, there may be orphaned deployments/other k8s and Istio resources in the cluster that would need to be cleaned up manually. Please contact EA to help reset these.
- Please contact EA on the [RefArch1.5 Chat Set - Troubleshooting Channel](#) if this happens to help diagnose the failure scenario and identify next steps. You can @mention [Aish](#), [Nate](#) & [Seshu](#) to get someone to look into it quickly.

Custom Configuration In Eks/Istio Clusters

Table of Contents

- [Custom Configuration In Eks/Istio Clusters](#)
 - [Background](#)
 - [How to add custom configuration aka. bring-your-own-yamls](#)
- [Ingress and Egress](#)
 - [Ingress](#)
 - [Egress](#)
- [Troubleshooting](#)

Background

- By default, teams will have read-only access to the Eks/Istio clusters and all changes/configuration to the cluster will be applied via automation (e.g., Jenkins pipelines for deployment process).
- Since RefArch1.5 infrastructure features are still in the pre-DevReady milestone phase as of PI12, any configuration needed to be applied to the cluster outside of the scope of these pipelines (e.g., ingress or egress routes) will be handled via a GitOps pattern through the [platform.infrastructure.tempkubeconfig](#) repo.

How to add custom configuration aka. bring-your-own-yamls

- Clone the [platform.infrastructure.tempkubeconfig](#) repo
- Switch to the branch representing the cluster where the change is to be applied e.g., EksI-VpcA-useast1-DeliveryDevQA-EASandbox
- Add the yaml file to the relevant folder using guidance below:
 - bootstrap -- for any configuration that is required to prime the cluster (e.g., emergent design needs for hydra or other infrastructure pipelines)
 - egress -- for any routing configuration to expose applications to services outside the mesh
 - ingress -- for any routing configuration to expose service methods via the istio-ingress endpoint
 - istio -- for any istio-specific configuration
 - tools -- for any tooling-specific configuration
- Make a PR on the branch (atleast 1 EA approval required)
- Once the PR is approved and merged into the branch, the configuration will be sync'd and applied to the cluster within 5 minutes automatically.

Ingress and Egress

Ingress and Egress to and from the Kubernetes/Istio service mesh is being built in PI12. For early access, we are providing the following process for adding Istio ingress and egress rules so you can interact with components outside the mesh.

Ingress

To expose methods within services via the istio-ingress endpoint (gateway url), you will need to update the ingress/global-ingress-vs.yaml file in the gitops repo [platform.infrastructure.tempkubeconfig](#) for the branch that maps to the cluster you are working on e.g., EksI-VpcA-useast1-DeliveryDevQA-EASandbox.

- Add a [match:] section with a uri-based math (prefix or exact match allowed) that will route to the destination service.
- Here's a sample file: [ingress/global-ingress-vs.yaml](#)

Egress

To expose applications to services that are outside the cluster, you will need the following kubernetes/istio resources applied to the cluster:

- [ServiceEntry](#)
- [VirtualService](#) *NOTE: VirtualService is required in addition to the ServiceEntry only for HTTPS connections outside the mesh.*

To add these resources you'll need to add a new yaml file in the gitops repo [platform.infrastructure.tempkubeconfig](#) for the branch that maps to the cluster you are working on e.g., EksI-VpcA-useast1-DeliveryDevQA-EASandbox. [platform.infrastructure.tempkubeconfig]

- Add k8s resource [kind: ServiceEntry] to allow access to hosts that are external to the mesh
- Add k8s resource [kind: VirtualService] to associate routes based on SNI values, to external hosts.

Here's a sample configuration for allowing egress traffic to reach eds-api hosts on both port 80 (http) and 443 (https): [egress-edsapi.yaml](#) in [EA-Sandbox cluster](#)

For more info, please see the Istio documentation on how to [Control Egress Traffic](#).

Troubleshooting

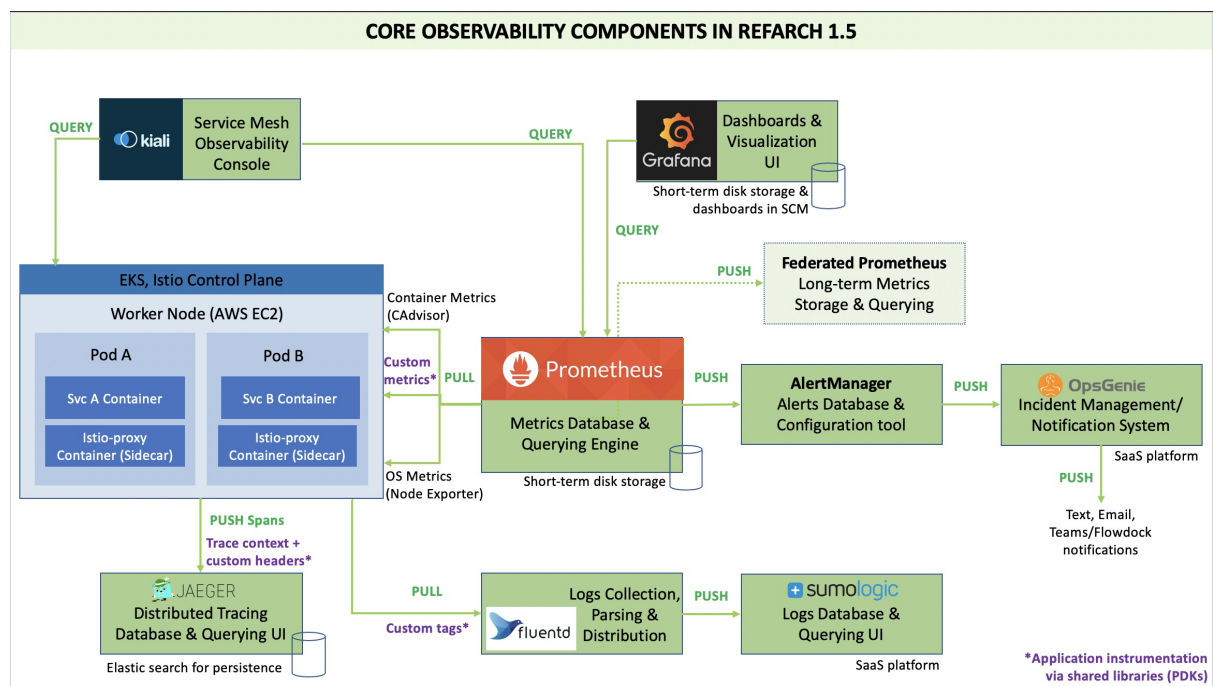
- Please reach out to EA on the [RefArch1.5 Chat Set - Troubleshooting Channel](#) to help diagnose any issues with this process. You can @mention [Aish](#), [Nate](#) & [Seshu](#) to get someone to look into it quickly.

Observability

Observability

- [Tools Ecosystem](#)
- [Tools and Roles](#)
- [Index page for monitoring tools](#)
- [Opsgenie](#)
- [Further reading](#)

Tools Ecosystem



Tools and Roles

Tool	Role	Instrumentation needed in services?
Fluentd	collecting, filtering, buffering, and outputting logs from containers to logging backend	integrate with refarch1.5 observability pdk for appending custom tags and standardization of logging format
Sumologic	managed logging platform - backend, visualization, analytics	none - integration provided by infrastructure pipeline
Prometheus	metrics aggregation service & backend, data source for alerts and dashboards	none for OOTB metrics integrate with refarch1.5 observability pdk & add annotations in code if custom metrics are needed
		none for OOTB dashboards

Grafana	metrics visualization via configurable dashboards	customization supported via modifications to json of default dashboards in source control
Alertmanager	alert management platform for prometheus handling deduplicating, grouping, and routing them to the receiver integration (e.g., opsgenie)	none for OOTB alerts customization supported
OpsGenie	alerting and on-call receiver/ management solution	none - integration provided by infrastructure pipeline
Jaeger	distributed tracing system, to provide end-to-end visibility and method-level insights into service requests	integrate with refarch1.5 observability pkg to propagate trace context. if it is service that doesn't make outbound calls then nothing is needed.
Kiali	visualize the topology of the service mesh in real time to identify bottlenecks and understand how data flows through the cluster. Includes realtime service dependency graph, distributed tracing integration, metrics integration, health checks, config viewing/validation, etc.	none - integration will be provided by infrastructure pipeline (pi12 item)

Index page for monitoring tools

Every cluster would come with an index page, which would contain links to the monitoring components for the EKS/Istio infrastructure and services.

Here's a sample of what this looks like presently for the `dev5` cluster:

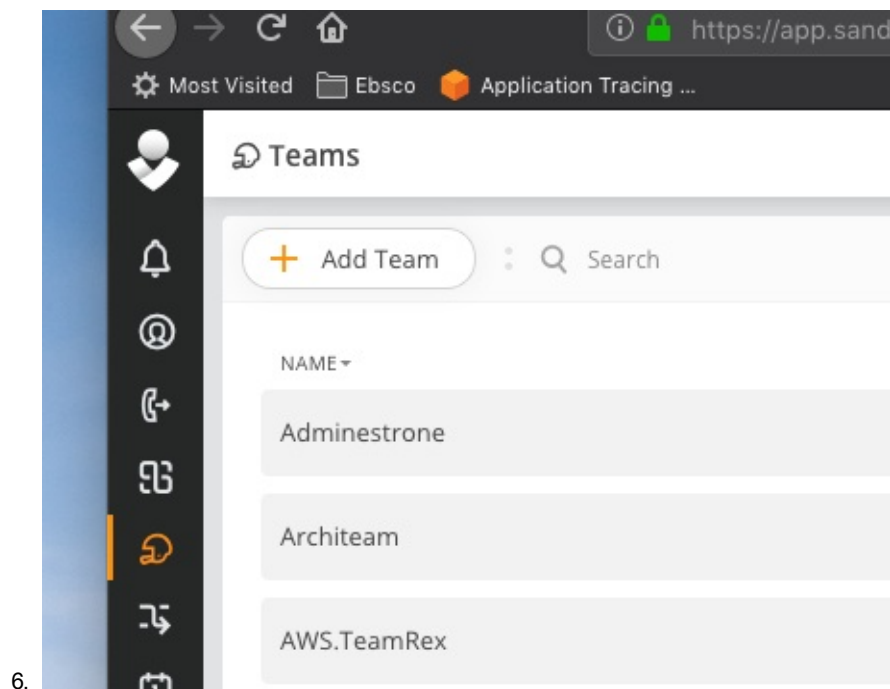
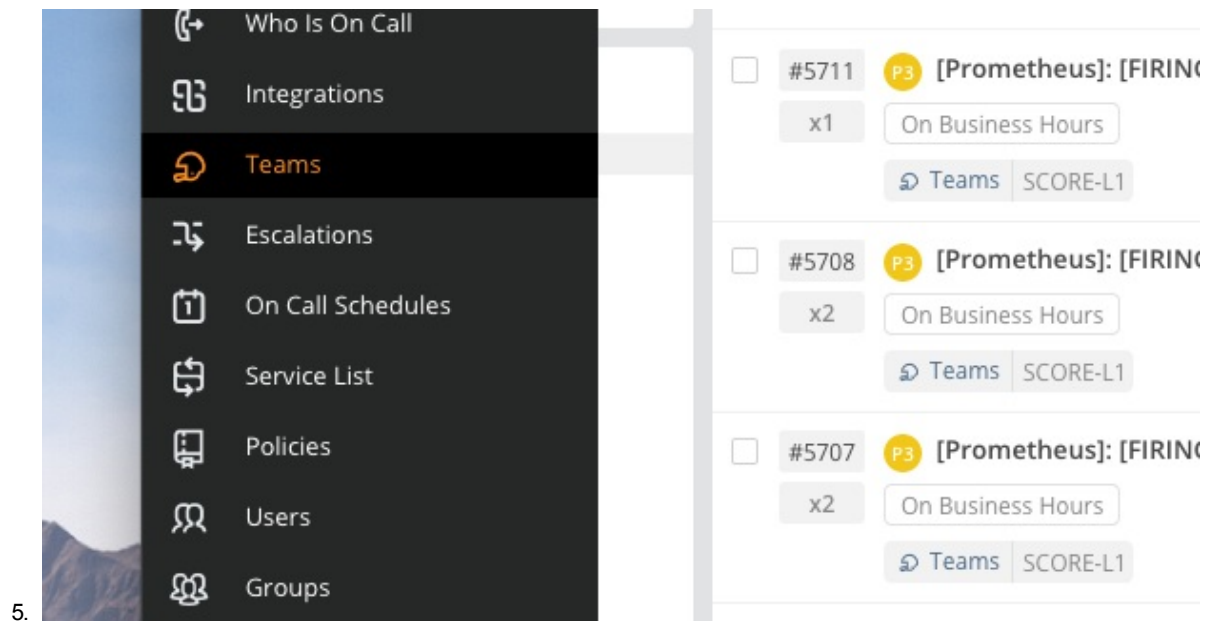
- [Links for ISTIO monitoring components](#)
- [Link to Kiali](#)
 - username/pwd: admin/admin
- [Link to login to Sumologic](#)

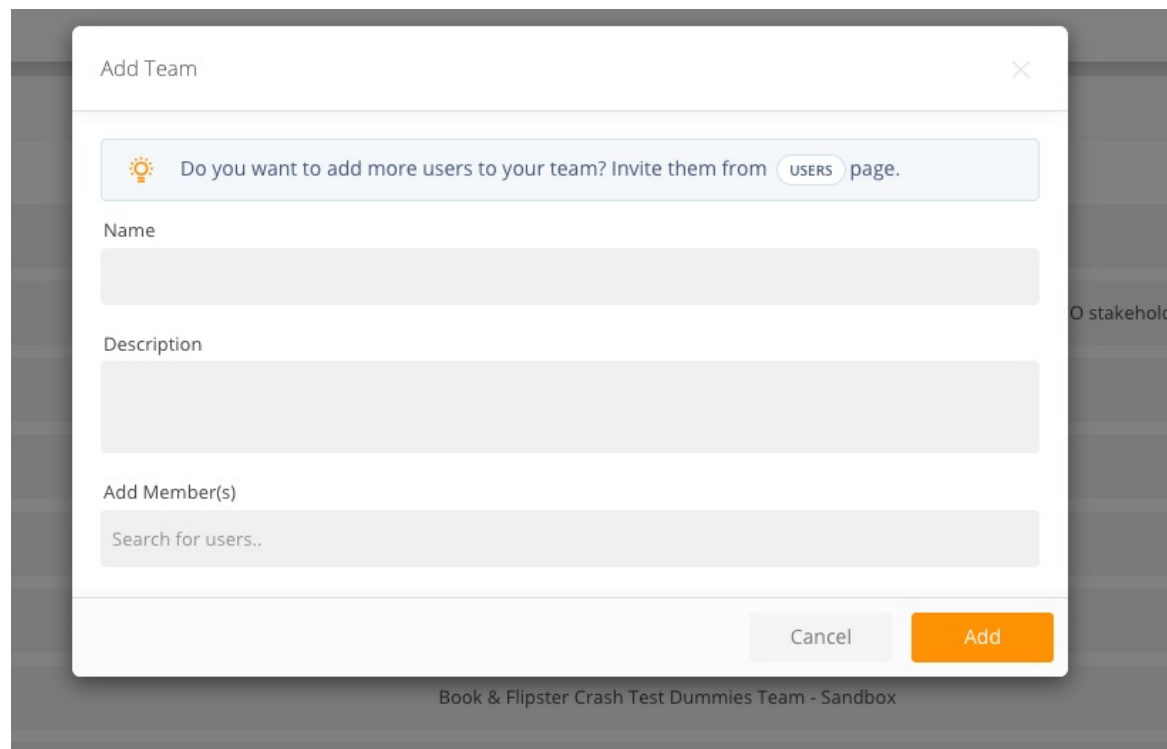
Opsgenie

Teams need to make sure they are added to Opsgenie to ensure delivery of alert notifications. Medusa will verify your team is configured in Opsgenie and will fail builds if your team is not found.

To add your team to OpsGenie:

1. Log into Opsgenie:
 - Sandbox: <https://app.sandbox.opsgenie.com>
 - Live: <https://app.opsgenie.com>
2. Select 'Teams' from the Opsgenie navigation menu
3. Click the 'Add Team' button
4. Fill in the details for your team and add your team members.
 - Your team name will need to match the team name in your applicationDefinition.yaml file.



7. 

Further reading

Please see the [Observability](#) page on confluence for more details.

Troubleshooting

- [Troubleshooting](#)
- [Github Autowire](#)
 - [Symptoms](#)
 - [Solution](#)
- [Crash Loop Detection](#)
 - [Symptoms](#)
 - [Solution](#)
 - [Walkthrough](#)
- [sudo: no tty present and no askpass program specified](#)
 - [Symptoms](#)
 - [Solution](#)
 - [Walkthrough](#)
- [Service is slow or crashing/restarting while running a performance or load test](#)
 - [Symptoms](#)
 - [Solution](#)
 - [Walkthrough:](#)

Github Autowire

Symptoms

Project or Repository does not appear in the Jenkins-Refarch15a Server despite having the jenkins-refarch15a label.

Solution

In the event that a repository does not show up in the Jenkins-RefArch15a server, please confirm:

1. <https://infrastructure.eis-platformlive.cloud> webhook exists in your repository. (Webhook configured to send only Label Events)
2. <https://jenkins-refarc15a-public.eis-platformlive.cloud> webhook exists in your repository (Webhook configured to send Pull Requests and Pushes)
3. jenkins-refarch15a label exists and is correctly spelt in your repository. DELETE label jenkins-refarch15a and then CREATE it again. Do not edit the existing label.

Crash Loop Detection

Crash loop detection is a feature within hydra that will fail your pipeline build if your service's container fails to start or crashes during startup.

Symptoms

You will see something similar to this in the Jenkins log for your project:

```

2018-10-15 14:33:51 Staging pods...
2018-10-15 14:33:52 Deployment ebsconext-edge-04cc528 found. Updating...
2018-10-15 14:33:52 Service ebsconext-edge found. Updating...
2018-10-15 14:33:52 ConfigMap ebsconext-edge-04cc528-config found. Updating...
2018-10-15 14:33:52 Horizontal Pod Autoscaler ebsconext-edge-hpa found. Updating...
2018-10-15 14:33:52 complete
2018-10-15 14:33:52 Running crashloop detection...
2018-10-15 14:33:53 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:33:58 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:03 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:08 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:13 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:18 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:23 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:28 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:33 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:38 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:44 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:49 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:54 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:34:59 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:35:04 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:35:09 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:35:14 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:35:19 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:35:24 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:35:30 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:35:35 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:35:40 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:35:45 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:35:50 Replica Status(Desired: 1, Current: None, Updated: 1, Available: None, Unavailable: 1)
2018-10-15 14:35:55 Deploy ebsconext-edge-04cc528 Not Successful: None pods available
2018-10-15 14:35:55 completed crashloop detection
2018-10-15 14:35:55 In a crashloop. Aborting...
2018-10-15 14:35:55 Rolling back version 04cc528...

```

Solution

First, you will need to ensure that your service is passing its health checks (Liveness Probe). If it is passing them then you will want to retrieve the logs from your service to figure out why it is not starting. You can get them by using the 'kubectl logs -n application' command.

Walkthrough

1. Rerun your Jenkins build and watch the console output for the line `Staging pods...`
2. Run `kubectl get pods -n application`
3. Find one of the failed pods by its name. The naming convention is `<Service Name>-<Version>-<UniqueID>` .
 - o An example pod name from the above jenkins log would be: `ebsconext-edge-04cc528-847ff5c57f-bfp7w`
4. Run `kubectl describe pod -n application <PodName>` to see if your pod's health checks are failing or are misconfigured.
 - o Example: `kubectl describe pod -n application ebsconext-edge-04cc528-847ff5c57f-bfp7w`
 - o If you see text indicating that a Liveness probe has failed then you must update your deployment manifest to include a Liveness check that returns a HTTP status 200 code [Liveness Checks](#)
 - o Failed Liveness Check Example: ``Warning Unhealthy 6m (x3 over 6m) kubelet, ip-10-25-230-114.ec2.internal Liveness probe failed: % Total % Received % Xferd Average Speed Time Time Time Current

Dload	Upload	Total	Spent	Left	Speed
-------	--------	-------	-------	------	-------

```
0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0curl: (7) Failed to connect to localhost port 8081: Connection refused
Warning Unhealthy 6m (x9 over 6m) kubelet, ip-10-25-230-114.ec2.internal Readiness probe failed: % Total
% Received % Xferd Average Speed Time Time Time Current
```

	Dload	Upload	Total	Spent	Left	Speed
--	-------	--------	-------	-------	------	-------

```
0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0curl: (7) Failed to connect to localhost port 8081: Connection refused
Warning BackOff 1m (x4 over 2m) kubelet, ip-10-25-230-114.ec2.internal Back-off restarting failed container
...
```

- Run `kubectl logs -n application <Pod Name> <Container Name>`
 - Example: `kubectl logs -n application ebsconext-edge-04cc528-847ff5c57f-bfp7w ebsconext-edge`
 - Use the application logs to diagnose why the container will not start.
- If there are no application logs or the output indicates a general kubernetes error then run `kubectl describe pod -n application <PodName>` and start a chat on the Troubleshooting RefArch 1.5 Chat Set including the output of this command.

sudo: no tty present and no askpass program specified

Symptoms

You receive an error in your Jenkins build indicating an error similar to `sudo: no tty present and no askpass program specified`.

```
++ id -u
+ sudo chown -R 501 .
sudo: no tty present and no askpass program specified
```

Solution

The use of sudo is no longer supported on Jenkins slaves. It must be removed from your Jenkinsfile in order for you build to run.

Walkthrough

Below is an example removing the use of sudo from a Java microservice Jenkinsfile.

Before:

```
#!/Groovy
@Library("platform.infrastructure.pipelinelibrary@refarch1.5-current") _

node(platformDefaults.getBuildNodeLabel()) {

    def ecrRegistry = platformDefaults.getDockerRegistry()

    stage("Checkout") {
        sh "sudo rm -rf build .gradle"
        deleteDir()
        checkout scm
    }
}
```

```

stage("Pull Build Image") {
    def credentials = platformDefaults.getCredentialsId()
    def region = platformDefaults.getRegion()

    docker.withRegistry("https://${ecrRegistry}", "ecr:${region}:${credentials}") {
        docker.image("platform/images/gradle").pull()
    }
}

stage('Build') {
    sh """
        docker run --rm -v \${HOME}/.gradle/gradle.properties:/home/gradle/.gradle/gradle.properties:ro -v \${WORKSPACE}/home/gradle:rw -u root \${ecrRegistry}/platform/images/gradle clean assemble test
        sudo chown -R \${id} -u) .
    """
}

dockerBuildPush()

```

After:

```

#!Groovy
@Library("platform.infrastructure.pipelinelibrary@refarch1.5-current") _

node(platformDefaults.getBuildNodeLabel()) {

    def ecrRegistry = platformDefaults.getDockerRegistry()

    stage("Checkout") {
        deleteDir()
        checkout scm
    }

    stage("Pull Build Image") {
        def credentials = platformDefaults.getCredentialsId()
        def region = platformDefaults.getRegion()

        docker.withRegistry("https://${ecrRegistry}", "ecr:${region}:${credentials}") {
            docker.image("platform/images/gradle-refarch15a").pull()
        }
    }

    stage('Build') {
        sh "docker run --rm -v \${HOME}/.gradle/gradle.properties:/home/gradle/.gradle/gradle.properties:ro -v \${WORKSPACE}/opt/ebSCO:rw \${ecrRegistry}/platform/images/gradle-refarch15a clean assemble test"
    }

    dockerBuildPush()
}

withNodeWrapper(platformDefaults.getDeployNodeLabel()) {
    hydraPreview deployEnv: "dev", instances: 1
    hydraDeploy deployEnv: "dev", instances: 1, namespace: 'application'
}

```

Service is slow or crashing/restarting while running a performance or load test

Symptoms

You are running a performance test, load test or generating a significant amount of traffic for your service and it is not performing well or is crashing and restarting often.

Solution

One possible cause is having improper resource requests and limits set for your pod.

- A resource request tells the kubernetes scheduler how much CPU and memory that your pod will normally use to run. If you underestimate your resource request then your pod may not receive the resources that it needs in order to operate properly and may be contending for resources with other pods. Additionally, when a kubernetes node is under resource pressure it will evict pods that are using more resources than they requested before other pods.
- A resource limit will limit the resources that your pod uses in order to protect the environment from excessive resource consumption. It is a best practice to have limits on all pods to help kubernetes balance resource appropriately. A CPU limit will throttle the maximum CPU consumption of your pod while a memory limit will restart you pod when exceeded.

CPU resources are measured in millicores (1000 millicores = 1 CPU core) and memory resources can be measured in units of memory.

You can see how kubernetes guarantees resources in the following [article](#).

There are 3 classes of resource QoS in kubernetes:

- **Best Effort:** If your pod has no resource requests or limits then kubernetes will make a best effort to give your pod resources but will not guarantee anything.
- **Burstable:** A burstable pod means that it has resource requests that are lower than the limits. Kubernetes will guarantee the resource requests and make a best effort to provide resources up to the resource limits.
- **Guaranteed:** A guaranteed pod means that it has both resource requests and limits and that the values for both are identical. Kubernetes will then guarantee those resources to that pod.

Walkthrough:

You can see if you are exceeding your resource requests or limits by querying Prometheus. In the future, this capability will be available in Grafana.

To see if you are exceeding your CPU or memory requests:

1. Go to [prometheus](#).
2. The following query will tell you the difference between how many millicores your service is using compared to the amount that it has requested. A negative value indicates using fewer resources than requested while a positive value indicates using more than requested.
 - `(sum(rate(container_cpu_usage_seconds_total{namespace="application", pod_name=~"ebsconext-ui.*"}[30m])) by (pod_name) - on (pod_name) label_replace(sum(kube_pod_container_resource_requests_cpu_cores{}) by (pod), "pod_name", "$1", "pod", "(.*)")) * 1000`
 - To change the query to look at your service replace the text `ebsconext-ui` in the field `pod_name=~".*ebsconext-ui.*"` to the name of your service.
 - To change the time range for the query (defaults to 30 minutes) change the `[30m]` value to the number of minutes that you desire.
 - Once complete click the execute button
3. The following query will tell you the difference between how many MB of memory your service has requested and how much it is using. A negative value indicates using fewer resources than requested while a positive value indicates using more than requested.
 - `(sum (avg_over_time(container_memory_usage_bytes{namespace="application", pod_name=~"ebsconext-ui.*"}[30m])) by (pod_name) - on (pod_name) label_replace(sum(kube_pod_container_resource_requests_memory_bytes{}) by (pod), "pod_name", "$1", "pod", "(.*)")) / 1024 / 1024`

- o To change the query to look at your service replace the text `ebsconext-ui` in the field `pod_name=~".*ebsconext-ui.*"` to the name of your service.
 - o To change the time range for the query (defaults to 30 minutes) change the `[30m]` value to the number of minutes that you desire.
- 4. Once you have the correct values from these queries you can update the Resources Policy section of your [deployment-manifest](#) with more accurate values.

Local Development and Debugging in RefArch 1.5

- [Local Development and Debugging in RefArch 1.5](#)
 - [Development Tools for RefArch 1.5](#)
 - [Common Environment Setup](#)
 - [Microservices](#)
 - [UI + edge](#)
 - [Debugging](#)
 - [Debugging Deployed Services](#)

Development Tools for RefArch 1.5

Common Environment Setup

- Python >= 2.7.15
- AWS Authentication
- `aws-iam-authenticator`
- `aws-adfs` should be setup to use `kubect1`.
 - [Detailed Instructions](#)
- Artifactory Access
- Kubernetes Config
- [Kubect1](#)
- Git
- [Docker latest](#) (optional)

Microservices

- [OpenJDK 11.x](#)
- Artifactory Authentication – `gradle.properties`

UI + edge

- [NodeJS 8.x LTS](#) (End of Octobose 2018 will switch to 10.x LTS)
- Artifactory Authentication - `npmrc`
- Npm >= 6
- Yeoman

Debugging

- Visual Studio Code with [Kubernetes plugin](#) (optional)
- [telepresence](#) (optional)

Debugging Deployed Services

Telepresense can be used to troubleshoot services deployed in Kubernetes cluster. Traffic to a POD can be hijacked and routed to developer machine (preview environment only).

For example:

```
telepresence --swap-deployment hello-world --run gradle bootRun
```

Above command will run `gradle bootRun` and forward traffic destined for `hello-world` to local machine.

Telepresence creates secure tunnel between developer machine and Kubernetes cluster. When running, developer can seamlessly resolve and access Kubernetes services

Useful scenarios:

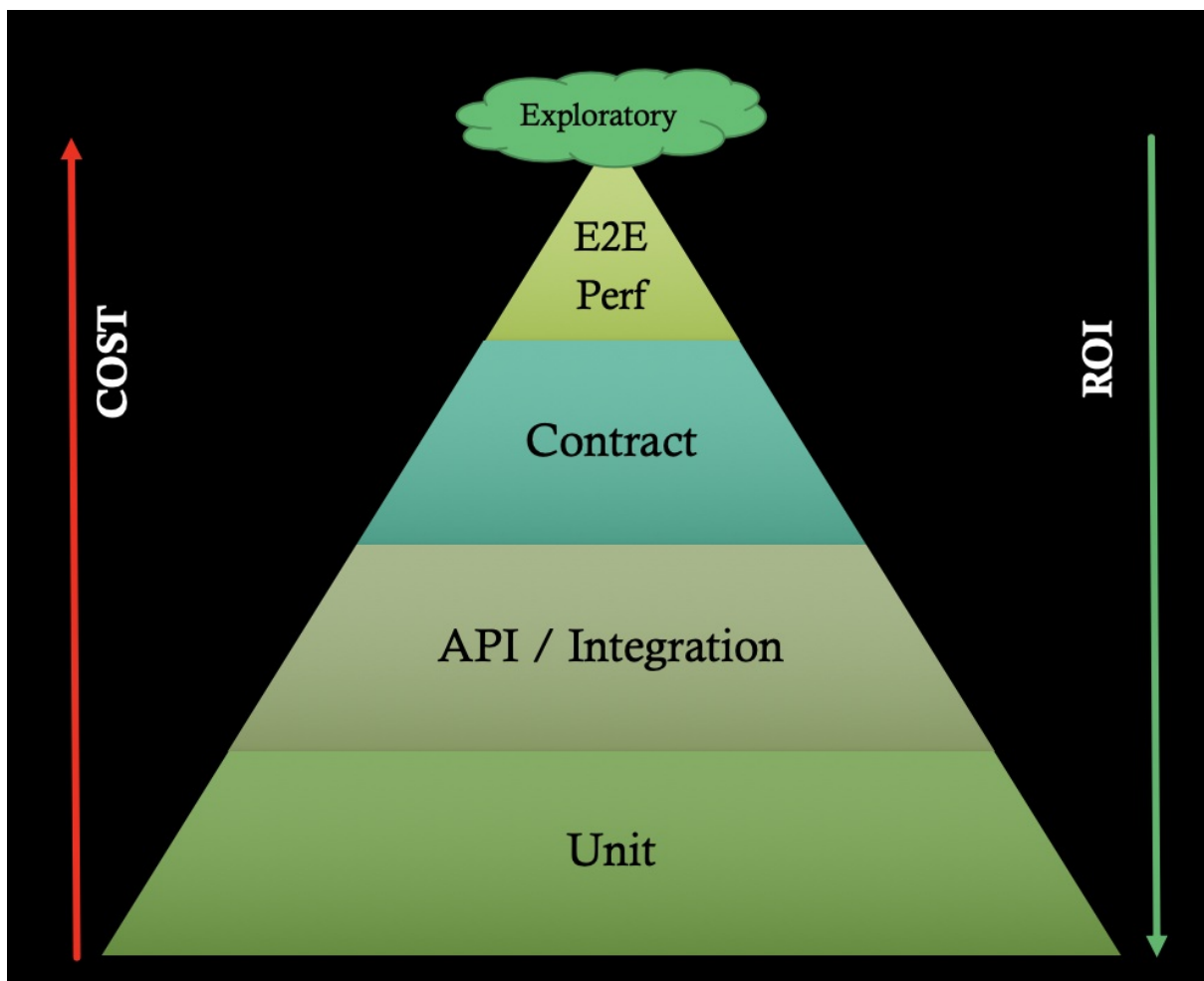
- [Debug a Kubernetes service locally](#)
- [Connect to a remote Kubernetes cluster](#)
- [Rapid development with Kubernetes](#)

Quality Gates Supported in the RefArch1.5 CI/CD Pipeline (Medusa)

- [Test Stages](#)
- [Guidance on Writing Tests For Automated Quality Gates](#)
 - [Unit Tests](#)
 - [Integration Tests](#)
 - [Consumer-driven Contract Test \(CDC\)](#)
 - [Performance Tests \(Server-side\)](#)
 - [Performance Tests \(Client-side\)](#)
 - [E2E Tests](#)

[Medusa](#) is the pipeline library which orchestrates RefArch 1.5 CI/CD stages. This guide captures the various quality gates/test stages that will be supported by the library.

Test Stages



Test Category	Execution Stage	Required?	Purpose	Role of test doubles	Network calls guidance
			Fast, low-level tests		

Unit Tests	CI / pre-deploy	Yes	to assert on the behavior of the code i.e., every model, class/controller, functions and view, in isolation.	see section below for guidance	No network calls made	<ul style="list-style-type: none"> • (u) •
Integration Tests	CI / pre-deploy	Yes	<p>Narrow integration tests that live at the boundary of your service, to validate component integration points. The intent is to write integration tests for all pieces of code where you either serialize or de-serialize data. some examples:</p> <ul style="list-style-type: none"> • Calls to your services' REST API • Reading from and writing to databases • Calling other application's APIs • Reading from and writing to queues • Writing to the filesystem 	see section below for guidance	Localhost only	<ul style="list-style-type: none"> • (u) •
Consumer-driven Contract Test (CDC)	CI / pre-deploy	For producer services that expose interfaces to other services only.	Services that provide interfaces (providers/producers) fetch and run CDC tests continuously to spot any breaking changes immediately. The verifier tests are auto-generated based on contracts expressed as DSL from the consumers.	see section below for guidance	Localhost only	
Static Code Analysis	CI / pre-deploy	Yes	<p>Code quality analysis to measure test coverage, run security scans, look for code smells, etc. Please see this yammer post for more information.</p> <ul style="list-style-type: none"> • Note: The quality profile for all projects is set to default to EBSCO QGate. 	N/A	N/A	

			If you need help changing this to a stricter profile for your project, please reach out to EA.			
E2E Tests	CD / post-deploy	Yes	Reliable workflow(s) to smoke test the "ends" that the system touches i.e., downstream calls only. e.g., middle->db, ui->edge->middle->db. This is meant to be a deployment qualification test to verify the container deployment worked and the application is up and able to receive requests and communicate with its dependencies.	see section below for guidance	Can access real services/external systems, databases, file systems, etc. over the network.	At the system level
Server-side Performance Tests	CD/ post-deploy	Required (implicit perf validation via Dark release stage during hydra deployment is an option, but will need to be validated by your system architect for viability. Also, this is not viable until refarch1.5 is live and receiving customer traffic.)	Validate that the service is able to meet the performance goals (relative and absolute assertions against response time under load, error rate)	see section below for guidance	Can access real services/external systems, databases, file systems, etc. over the network	At the transaction level
Client-side Performance Tests	CD/ post-deploy	Only for UX services	To opportunities around improving the end-user experience by identifying optimizations that can be applied to a UI page (mobile or desktop experience) to improve TTFB, render time, DOM load time (e.g., leveraging browser caching, enabling compression,	see section below for guidance	Can access real services/external systems, databases, file systems, etc. over the network	At the client level

			minifying html/css, optimizing rendering experience by inlining JS or making external requests async, etc.)		
--	--	--	--	--	--

Guidance on Writing Tests For Automated Quality Gates

Unit Tests

Use of test doubles

Aggressively mock all dependencies, including:

- external service dependencies (i.e. anything that crosses a process/network boundary)
- external libraries
- classes/ functions in the same project

Integration Tests

Use of test doubles

Aim to run your external dependencies locally i.e., mock/stub dependencies not owned by the service, like:

- in-process test doubles to test integration with external systems e.g., in-memory embedded databases, local MySQL database, local ext4 filesystem
- out-of-process stubs to replace an external service (recommended to use the stubs published by the producer service during the contract testing stage)

Consumer-driven Contract Test (CDC)

Use of test doubles

The framework provides 2 frameworks to execute contract verifier tests:

- For gradle/maven based spring boot services, use MockMvc framework to mock dependencies.
- For all others (e.g., nodejs or .NET), use stub servers to replace calls to external dependencies if CDC tests are run in non-MVC mode (i.e., explicit-mode). These are out-of-process test-doubles that help mock external service calls using reliable stubs published from the downstream producer's contract test stage.

Performance Tests (Server-side)

Use of test doubles

Recommendation is to write "narrow perf tests" to isolate measurements to service-under-test by using stubs to replace downstream dependencies. Watch this space for updates -- more to come when we have examples/reference implementation for these scenarios).

JMX Test Attributes

- Please refer to the Medusa guide on [Performance Tests](#) for specifications around where the tests are discovered from and more about the performanceDefinition.yaml file.
- The `TestProperties` section in the performanceDefinition file helps define the attributes of the jmeter test (`jmx` test plan) when Medusa orchestrates the test as an automated quality gate during the CD-phase of the pipeline.
- In order for these properties to take effect, a matching JMeter property (referenced using `${__P(duration,60)}`) in

test plan, and passed into the command-line invocation of the jmx file using `-j` argument) should be present in the test plan.

- e.g., if `duration` is provided under the `TestProperties` section, the `jmx` test plan should implement and use the `-jduration` property.
- Recommendations for performance tests in the pipeline:
 - Provide `duration` in the `TestProperties` section, to limit the tests to a fixed duration.
 - Minimum duration should be at least >1 minute (i.e., 2x the prometheus scrape interval, which defaults to 30s), and maximum duration should be well under the <15 minutes to stay within the `hydra` timeout setting for test jobs.
 - Recommended duration for tests is 5-10 minutes i.e., 300 seconds - 600 seconds.
 - Provide `throughput`, `threads`, `rampup` (optional) attributes in the `jmx` test plan to help drive the load levels (`rps` or `requests per second`) for your tests.
 - Recommendation is to use steady load (throughput) during these tests, instead of variable load since these are short-duration tests and variability in load levels can lead to inconsistent measurements on aggregate calculations of response time.
 - Ramp up/down can be used, but the validation currently does not offset these windows from calculating 95th percentile response time. This will be a future enhancement added to the platform (tentatively in PI14).
 - The `Constant Throughput Timer` element can be used to throttle/vary the request rates for the `HTTP Request` elements in the test plan. When this is applied at the `Thread Group` level it applies to all `HTTP Request` elements under that group, but you could use the timer within an `HTTP Request` element to apply a variable rate and override the parent-rate for specific requests within a thread group.

Performance Tests (Client-side)

Recommendation is to follow this reference implementation from EA: [platform.infrastructure.docker-lighthouse](#).

E2E Tests

Typically, no test doubles/mocking recommended. The purpose of these tests is to test integration with external services, persistence, startup logic, authentication workflows, network configuration and remote request/response routing with dependencies.

For UX svcs: Validate limited user-critical workflows, to test integration with downstream dependencies. Model tests around users of the system and the journeys those users make through the system (business cases).

As an exception, mocking could be considered for async back-end processes and other flaky external services that might be part of the e2e workflow for the service. Refer guidance from Martin Fowler "If a particular external service or GUI is a major cause of flakiness in the test suite, it can help to redefine the test boundary to exclude the component. In this case, total end-to-end coverage is traded in favour of reliability in the suite. This is acceptable as long as other forms of testing verify the flaky component using different means."

Please see this page for more guidance around e2e tests: [E2E Testing for Microservices](#)

Code Examples

- [Code Examples](#)
 - [Conventions](#)
 - [Middle Service](#)
 - [Edge Service](#)
 - [Calling EBSCO Next API](#)
 - [Secret Management](#)
 - [Queues](#)
 - [Calling a Database](#)
 - [Propagating headers for distributed tracing](#)
 - [Pushing custom metrics to Prometheus](#)

Conventions

- **SERVICE PORT**
 - All services will be exposed on port **8080** (java, nodejs edge & ui).
- **SERVICE NAME**
 - Service names should not include dots, as this will interfere with the DNS-based resolution in the Kubernetes cluster.
 - Must be a DNS 952 label (at most 24 characters, matching regex `a-z?`): e.g. "my-name"
- **SERVICE DISCOVERY**
 - Services will use Kubernetes DNS resolution to discover its dependencies.
 - All services are assigned a DNS A record for a name of the form **my-svc.my-namespace.svc.cluster.local**. This resolves to the cluster IP (or to the set of IPs of the pods selected by the Service in the case of headless services) of the Service.
 - Example:
 - To discover a service called 'bookProvider' running on port 8080 in the cluster in the 'default' namespace, the consumer service (bookConsumer) will use the endpoint <http://bookprovider:8080> as the baseUrl.
 - To discover a service called 'bookProvider' running on port 8080 in the cluster in the 'application' namespace, the consumer service (bookConsumer) will use the endpoint <http://bookprovider.application.svc.cluster.local:8080> as the baseUrl.

Middle Service

[ea.shared.simplmiddle](#) - [refarch-1.5-with-springboot-2](#) branch

[platform.training.bookprovider](#) - [istio](#) branch

Edge Service

[ea.shared.simpleedge](#) - [refarch-1.5](#) branch

Calling EBSCO Next API

See auth workflows in common/models/search.js of this repo: [discover.edspoc.edgeapi](#)

Secret Management

- See /config/secrets/edsapi-secrets.json and /server/lib/auth.js for how secrets are propagated in this repo: [discover.edspoc.edgeapi](#)
- Long-term this will be handled via Hashicorp Vault and the docs will be updated to reflect that.

Queues

TBD

Calling a Database

TBD

Propagating headers for distributed tracing

With Node.js/Express middleware

```
// middleware.js
const axios = require("axios");

const headersToPropagate = [
  "x-request-id",
  "x-b3-traceid",
  "x-b3-spanid",
  "x-b3-parentspanid",
  "x-b3-sampled",
  "x-b3-flags",
  "x-ot-span-context"
];

module.exports = (req, res, next) => {
  req.axios = axios.create({
    headers: headersToPropagate.reduce((acc, header) => {
      const val = req.get(header);
      if (val) {
        acc[header] = val;
      }
      return acc;
    }, {})
  });
  next();
};
```

```
// server.js
const express = require("express");
const app = express();
app.use(require("./middleware"));

app.get("/api/foo", (req, res) => {
  // Make an outbound call to mid-tier using the Axios instance provided by middleware
  req.axios.get("http://someservice:8080")
    .then(({data}) => res.json(data))
    .catch(err => res.status(500).json({ message: "Hit an error!" }));
});
```

```
});
```

Pushing custom metrics to Prometheus

TBD

RefArch 1.5 Platform Release Notes

NOTE: Please refer to the [Troubleshooting Guide](#) for the resolution of common issues.

- [RefArch 1.5 Platform Release Notes](#)
 - [PI13 I5 Platform Upgrade](#)
 - [OVERVIEW](#)
 - [DEPRECATIONS/CHANGES](#)
 - [New cluster](#)
 - [JSB application.yml changes](#)
 - [Node PDK upgrade](#)
 - [New Jenkins Server \(refarch15e\) for services that use Medusa ONLY](#)
 - [New Secrets Retrieval Mechanism for using VAULT](#)
 - [NEW FEATURES](#)
 - [Gradle plugins and 5.2.1 upgrade](#)
 - [Put secrets in SSM Parameter Store using vault UI](#)
 - [BUG FIXES](#)
 - [NPM audit](#)

PI13 I5 Platform Upgrade

To review release notes for the previous platform upgrades, please see [archived-release-notes](#).

OVERVIEW

- **New cluster:** EksJ becomes EksK
- **JSB application.yml:** All JSB services should delete indicated sections from config
- **JSB Gradle upgrade and plugins changes:** All JSB services should make indicated dependency changes
- **Node PDK upgrade:** All Node services should update to latest version of the PDK
- **New Jenkins server** [jenkins-refarch15e](#) for medusa-based projects.
- **Baseline Alerts:** All services will now have baseline alerts configured

DEPRECATIONS/CHANGES

New cluster

`EksK-VpcA-useast1-DeliveryDevQA-Dev6` . Kube config file available [here](#)

Monitoring tools page: <http://monitoring.eksk-useast1.eks.eis-deliverydevqa.cloud>

JSB application.yml changes

JSB repositories should have several items removed from `src/main/resources/application.yml` . A large portion of these settings are now managed by the PDK. Others are no longer in use. If a block no longer contains items after removal, remove the block as well. (This may be the case for `server` , `info` , `logging` and `management` .)

Remove the following configuration from `application.yml` :

```
spring:
  application:
    name: platform.training.example # your name
```

```

main:
  banner-mode: "off"

- server:
- port: 8080
-
application:
  team: ea          # your team
  market: training # your market
  domain: platform # your domain
- subenv: ${APP_SUB_ENV:subenv}
- environment: ${AWS_ENV:environment}
- containerId: ${HOSTNAME:container}
- transactionIdHeader: "TRANSACTION_ID"
-
- info:
- title: ${spring.application.name}
- market: ${application.market}
- domain: ${application.domain}
- version: ${APP_VERSION:unknown}
- environment: ${AWS_ENV:environment}
- containerId: ${HOSTNAME:container}
-
- logging:
- level:
-   root: INFO
-
- management:
- server:
-   port: 8081
- endpoints:
-   web:
-     exposure:
-       include: env,health,info,metrics,prometheus,threaddump
-     base-path: /admin
-     path-mapping:
-       prometheus: prometheus
- endpoint:
-   metrics:
-     enabled: true
-   prometheus:
-     enabled: true
-   metrics:
-     export:
-       prometheus:
-         enabled: true

```

Node PDK upgrade

An important patch release has been made available and all Node services should upgrade; this is a non-breaking change that will align services to changes in platform-level metrics collection, so all that is required is to do an npm install:

```
npm i --save platform.shared.node-express-pdk@2.0.2
```

Please make sure to commit the resulting changes in both the package.json and package-lock.json files to SCM.

New Jenkins Server (refarch15e) for services that use Medusa ONLY

NOTE: Jenkins server jenkins-refarch15d will be decommissioned at the end of I6, if your project is on jenkins-refarch15d, it needs to be migrated to jenkins-refarch15e

- If you are **not** using / migrating to [Medusa](#), you should continue to use the `refarch15a` Jenkins instance & label
- The new Jenkins server is located at: <https://jenkins-refarch15e.eis-platformlive.cloud>

- To move your job from <https://jenkins-refarch15d.eis-platformlive.cloud> to <https://jenkins-refarch15e.eis-platformlive.cloud>
 - Remove the `jenkins-refarch15d` label from your repo
 - Confirm job was removed from <https://jenkins-refarch15d.eis-platformlive.cloud>
 - Add a new label `jenkins-refarch15e` to your repo
 - Confirm job was added from <https://jenkins-refarch15e.eis-platformlive.cloud>
 - **When switching labels, delete the old label and create a new label with 'jenkins-refarch15e'. Do not edit the existing label**
- The only library supported in `jenkins-refarch15e` is [Medusa](#)

New Secrets Retrieval Mechanism for using VAULT

- You no longer have to put `config.hcl` or `template.ctmpl` file in your repo.
- Application code no longer needs to place above files in config directory.
 - This is an automated step
- You now add a `secrets-manifest.yaml` to the `/secrets/` directory in your repository. [Sample-Manifest Documentation](#)
- You no longer have to create multiple files to provision resources in Vault via `platform.infrastructure.vault-resources`.
- To Provision Resources create a PR in to Resources Directory in [Vault-Init](#)
- [Sample-Provision-Manifest](#)

NEW FEATURES

Gradle plugins and 5.2.1 upgrade

JSB repositories that use Medusa should make the following changes:

- Add two new plugins, `com.github.ben-manes.versions` and `com.pasam.gradle.buildinfo`, to `build.gradle` and update the versions of existing plugins. The result should contain the following:

```
plugins {
    // ...
    id 'com.github.ben-manes.versions' version '0.21.0'
    id 'com.pasam.gradle.buildinfo' version '0.1.3'
    id 'com.jfrog.artifactory' version '4.9.3'
    id 'org.owasp.dependencycheck' version '4.0.2'
    id 'org.sonarqube' version '2.7'
    id 'org.springframework.boot' version "2.1.3.RELEASE"
    id 'spring-cloud-contract' version "2.1.1.RELEASE"
}
```

- Update versions of the following dependencies in the `ext` section of `build.gradle` :

```
ext {
    // ...
    // Compile dependency versions
    amazonAWSJavaSDKVersion = '1.11.519'
    lombokVersion = '1.18.6'
    springBootVersion = '2.1.3.RELEASE'
    springCloudContractVersion = '2.1.1.RELEASE'
    swaggerVersion = '2.9.2'

    // Test dependency versions
    junitJupiterVersion = '5.4.1'
    junitLauncherVersion = '1.4.1'
    mockitoVersion = '2.23.4'
```

```
}
```

- Update the Jacoco tool version in `build.gradle` :

```
jacoco {  
    toolVersion = '0.8.3'  
}
```

- Add the following test runtime for `junit-platform-commons` to the `dependencies` section of `build.gradle` :

```
dependencies {  
    // Tests  
    // ...  
    testRuntime group: 'org.junit.platform', name: 'junit-platform-commons', version: "${junitLauncherVersion}"  
}
```

- Add `mavenLocal()` to `settings.gradle` :

```
pluginManagement {  
    repositories {  
        mavenLocal()  
        // ...  
    }  
}
```

- Run the following command to upgrade gradle wrapper to version 5.2.1:
 - `./gradlew wrapper --gradle-version 5.2.1 --distribution-type all`
- Stage and commit all the changes that includes the following files:
 - `gradle/wrapper/gradle-wrapper.jar`
 - `gradle/wrapper/gradle-wrapper.properties`
 - `gradlew`
 - `gradlew.bat`
 - `build.gradle`
 - `settings.gradle`

Put secrets in SSM Parameter Store using vault UI

- Select users can use VaultUI to put secrets in the AWS SSM Parameter store.
- [Using Vault UI to put secrets into AWS SSM Parameter Store](#)

Baseline Alerts

See [Alerting](#) guide

BUG FIXES

NPM audit

An intermittent failure had been observed in Jenkins during audit, breaking builds. Thanks to :star: [@elliottmrodriguez](#) :star:, the NPM audit helper now has a retry mechanism and improved error messaging.

Projected Release Notes

FAQs

FAQs

- [Windows](#)
- [Java](#)
- [Gradle](#)
- [Eclipse](#)
- [NodeJS / npm](#)
- [AWS authentication and tooling](#)
 - [Setup aws-adfs](#)
 - [Setup aws-iam-authenticator](#)
 - [Setup kube config](#)
 - [Setup kubectl](#)
- [Accessing services](#)
- [Dev tools](#)

Windows

- Windows 10 is preferred OS
- Make sure `HOME` environment variable is set
 - Open command prompt
 - `echo %HOME%` should display your home directory path

Java

- **DO NOT** install Oracle JDK
- Install [OpenJDK 11.x](#)
 - [macOS instructions](#)
 - [Windows instructions](#)
- All RefArch 1.5 development must use JDK 11.x or current LTS
- `java -version` should print something like the following:
 - `openjdk version "11" ...`

Gradle

- It is not necessary to install gradle. And gradle tasks should not be invoked using `gradle`.
- Use gradle wrapper to install gradle and run tasks. Examples:
 - `.\gradlew.bat clean assemble`
 - `./gradlew clean assemble`
- Java based microservices must use Gradle version `4.10.2`. `gradle/wrapper/gradle-wrapper.properties` must have:
 - `distributionUrl=https\://services.gradle.org/distributions/gradle-4.10.2-bin.zip`
- Gradle properties must be configured with artifactory information:
 - Windows: `%HOME%\gradle\gradle.properties`
 - macOS: `$HOME/.gradle/gradle.properties`
 - See [Java](#) section at the bottom of this [link](#)

- `artifactory_user` must be set to `<user>@corp.epnet.com` (change `<user>`)
- `artifactory_password` must be set to your personal API key from Artifactory
- `artifactory_contextUrl` must be set to `https://eis.jfrog.io/eis`

Eclipse

- Install Eclipse IDE for Java Developers Version: `2018-09 (4.9.0)` or better
- If necessary install [Java 11 Support](#) extension for Eclipse

NodeJS / npm

- Install NodeJS LTS version. Currently LTS is `8.12.x` . End of October 2018, NodeJS `10.x` will be LTS
- Install npm `6.4` or better (`npm install npm@latest -g`)
- npm configuration must be updated to point to artifactory:
 - Windows: `%HOME%\npmrc`
 - macOS: `$HOME/.npmrc`
- See `JavaScript` section in this [link](#). Once configured `.npmrc` must contain:
 - `always-auth` set to `true`
 - `email` set to `<user>@corp.epnet.com` (change `<user>`)
 - `registry` set to `https://eis.jfrog.io/eis/api/npm/npm`
 - `_auth` token **MUST NOT** be set to API key. See `Steps to generate _auth` section in confluence link. Run the `curl` command with user and API token and the output from `curl` command must be set to `_auth` value

AWS authentication and tooling

Setup aws-adfs

- [Link to instructions on confluence to setup aws-adfs](#)
- **NOTE:** Omit the `--profile` argument when setting this up, else the `AWS_PROFILE` env variable will need to be setup to point to the correct profile context for kubectl commands to work.
 - Example:
 - `aws-adfs login --region=us-east-1 --adfs-host=fsx.ebsco.com` will not require `AWS_PROFILE` env variable to be set
 - `aws-adfs login --profile=eis-deliverydevqa --region=us-east-1 --adfs-host=fsx.ebsco.com` will require the `AWS_PROFILE` env variable to be set to `eis-deliverydevqa`
 - For MacOS: `export AWS_PROFILE=eis-deliverydevqa`
 - For Windows: `set AWS_PROFILE=eis-deliverydevqa`

Setup aws-iam-authenticator

<https://docs.aws.amazon.com/eks/latest/userguide/configure-kubectl.html>

Setup kube config

- download or copy [kube config for Dev3 cluster](#)
- `config` file should be located under
 - `~/.kube` for MacOS
 - `$HOME/.kube` for Windows

- verify if the current context is pointing to the correct cluster/environment by using the following commands:
 - `kubectl config current-context`
 - `kubectl config view`

Setup kubectl

- <https://docs.aws.amazon.com/eks/latest/userguide/configure-kubectl.html>
- `kubectl` version must be `1.10` or better. Older versions will not work. Run `kubectl version` to check version

Accessing services

- Access services using the service URL:
 - Run `kubectl get virtualservices` and identify the service name
 - Access the service as: `http://<service name>.<cluster domain>/<context path>/<endpoint>` .
 - Example: `http://arch-ea-platform-training-simplemiddle-tommitchell.eksi-useast1.eks.eis-deliverydevqa.cloud/tommitchell/search?q=moon`

Dev tools

Nice-to-have dev tools:

- Visual Studio Code
 - add Kubernetes plugin
- [Telepresence](#)

Markdown: Syntax

- [Overview](#)
 - [Philosophy](#)
 - [Inline HTML](#)
 - [Automatic Escaping for Special Characters](#)
- [Block Elements](#)
 - [Paragraphs and Line Breaks](#)
 - [Headers](#)
 - [Blockquotes](#)
 - [Lists](#)
 - [Code Blocks](#)
 - [Horizontal Rules](#)
- [Span Elements](#)
 - [Links](#)
 - [Emphasis](#)
 - [Code](#)
 - [Images](#)
- [Miscellaneous](#)
 - [Backslash Escapes](#)
 - [Automatic Links](#)

Note: This document is itself written using Markdown; you can [see the source for it by adding '.text' to the URL](#).

Overview

Philosophy

Markdown is intended to be as easy-to-read and easy-to-write as is feasible.

Readability, however, is emphasized above all else. A Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions. While Markdown's syntax has been influenced by several existing text-to-HTML filters -- including [Setext](#), [atx](#), [Textile](#), [reStructuredText](#), [Grutatext](#), and [EtText](#) -- the single biggest source of inspiration for Markdown's syntax is the format of plain text email.

Block Elements

Paragraphs and Line Breaks

A paragraph is simply one or more consecutive lines of text, separated by one or more blank lines. (A blank line is any line that looks like a blank line -- a line containing nothing but spaces or tabs is considered blank.) Normal paragraphs should not be indented with spaces or tabs.

The implication of the "one or more consecutive lines of text" rule is that Markdown supports "hard-wrapped" text paragraphs. This differs significantly from most other text-to-HTML formatters (including Movable Type's "Convert Line Breaks" option) which translate every line break character in a paragraph into a `
` tag.

When you *do* want to insert a `
` break tag using Markdown, you end a line with two or more spaces, then type return.

Headers

Markdown supports two styles of headers, [Setext] [1] and [atx] [2].

Optionally, you may "close" atx-style headers. This is purely cosmetic -- you can use this if you think it looks better. The closing hashes don't even need to match the number of hashes used to open the header. (The number of opening hashes determines the header level.)

Blockquotes

Markdown uses email-style `>` characters for blockquoting. If you're familiar with quoting passages of text in an email message, then you know how to create a blockquote in Markdown. It looks best if you hard wrap the text and put a `>` before every line:

```
This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.
```

```
Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetur libero luctus adipiscing.
```

Markdown allows you to be lazy and only put the `>` before the first line of a hard-wrapped paragraph:

```
This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.
```

```
Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetur libero luctus adipiscing.
```

Blockquotes can be nested (i.e. a blockquote-in-a-blockquote) by adding additional levels of `>` :

```
This is the first level of quoting.
```

```
    This is nested blockquote.
```

```
Back to the first level.
```

Blockquotes can contain other Markdown elements, including headers, lists, and code blocks:

This is a header.

1. This is the first list item.
2. This is the second list item.

Here's some example code:

```
return shell_exec("echo $input | $markdown_script");
```

Any decent text editor should make email-style quoting easy. For example, with BBEdit, you can make a selection and choose Increase Quote Level from the Text menu.

Lists

Markdown supports ordered (numbered) and unordered (bulleted) lists.

Unordered lists use asterisks, pluses, and hyphens -- interchangeably -- as list markers:

- Red
- Green
- Blue

is equivalent to:

- Red
- Green
- Blue

and:

- Red
- Green
- Blue

Ordered lists use numbers followed by periods:

1. Bird
2. McHale
3. Parish

It's important to note that the actual numbers you use to mark the list have no effect on the HTML output Markdown produces. The HTML Markdown produces from the above list is:

If you instead wrote the list in Markdown like this:

1. Bird
2. McHale
3. Parish

or even:

1. Bird
2. McHale
3. Parish

you'd get the exact same HTML output. The point is, if you want to, you can use ordinal numbers in your ordered Markdown lists, so that the numbers in your source match the numbers in your published HTML. But if you want to be lazy, you don't have to.

To make lists look nice, you can wrap items with hanging indents:

- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.
- Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetur libero luctus adipiscing.

But if you want to be lazy, you don't have to:

- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.
- Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetur libero luctus adipiscing.

List items may consist of multiple paragraphs. Each subsequent paragraph in a list item must be indented by either 4 spaces or one tab:

1. This is a list item with two paragraphs. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus.

Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus. Donec sit amet nisl. Aliquam semper ipsum sit amet velit.

2. Suspendisse id sem consectetur libero luctus adipiscing.

It looks nice if you indent every line of the subsequent paragraphs, but here again, Markdown will allow you to be lazy:

- This is a list item with two paragraphs.

This is the second paragraph in the list item. You're only required to indent the first line. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

- Another item in the same list.

To put a blockquote within a list item, the blockquote's `>` delimiters need to be indented:

- A list item with a blockquote:

This is a blockquote inside a list item.

To put a code block within a list item, the code block needs to be indented *twice* -- 8 spaces or two tabs:

- A list item with a code block:

```
<code goes here>
```

Code Blocks

Pre-formatted code blocks are used for writing about programming or markup source code. Rather than forming normal paragraphs, the lines of a code block are interpreted literally. Markdown wraps a code block in both `<pre>` and `<code>` tags.

To produce a code block in Markdown, simply indent every line of the block by at least 4 spaces or 1 tab.

This is a normal paragraph:

```
This is a code block.
```

Here is an example of AppleScript:

```
tell application "Foo"
  beep
end tell
```

A code block continues until it reaches a line that is not indented (or the end of the article).

Within a code block, ampersands (`&`) and angle brackets (`<` and `>`) are automatically converted into HTML entities. This makes it very easy to include example HTML source code using Markdown -- just paste it and indent it, and Markdown will handle the hassle of encoding the ampersands and angle brackets. For example, this:

```
<div class="footer">
  &copy; 2004 Foo Corporation
</div>
```


Regular Markdown syntax is not processed within code blocks. E.g., asterisks are just literal asterisks within a code block. This means it's also easy to use Markdown to write about Markdown's own syntax.

```
tell application "Foo"
    beep
end tell
```

Span Elements

Links

Markdown supports two style of links: *inline* and *reference*.

In both styles, the link text is delimited by [square brackets].

To create an inline link, use a set of regular parentheses immediately after the link text's closing square bracket. Inside the parentheses, put the URL where you want the link to point, along with an *optional* title for the link, surrounded in quotes. For example:

This is [an example](#) inline link.

[This link](#) has no title attribute.

Emphasis

Markdown treats asterisks (`*`) and underscores (`_`) as indicators of emphasis. Text wrapped with one `*` or `_` will be wrapped with an HTML `` tag; double `*` 's or `_` 's will be wrapped with an HTML `` tag. E.g., this input:

single asterisks

single underscores

double asterisks

double underscores

Code

To indicate a span of code, wrap it with backtick quotes (```). Unlike a pre-formatted code block, a code span indicates code within a normal paragraph. For example:

Use the `printf()` function.

```
// Test.java
public class HelloWorld {
    public static void main(String[] args) {
        // Prints "Hello, World" to the terminal window.
        System.out.println("Hello, World");
    }
}
```

No video url found - vimeo and youtube supported

