Chapter 1 – Key CI/CD/Jenkins Concepts 5

Continuous Delivery/Continuous Integration Concepts 5

Define continuous integration, continuous delivery, continuous deployment 5

Difference between CI and CD 5

Stages of CI and CD 5

Continuous delivery versus continuous deployment 5

Jobs 6

What are jobs in Jenkins? 6

Types of jobs 6

Scope of jobs 6

Builds 6

What are builds in Jenkins? 6

What are build steps, triggers, artifacts, and repositories? 6

Build tools configuration 6

Source Code Management 6

What are source code management systems and how are they used? 6

Cloud-based SCMs 6

Jenkins changelogs 6

Incremental updates v clean check out 7

Checking in code 7

Infrastructure-as-Code 7

Branch and Merge Strategies 7

Testing 7

Benefits of testing with Jenkins 7

Define unit test, smoke test, acceptance test, automated verification/functional tests 7

Notifications 7

Types of notifications in Jenkins 7

Importance of notifications 7

Distributed Builds 7

What are distributed builds? 7

Functions of masters and slaves 7

Plugins 8

What are plugins? 8

What is the plugin manager? 8

Jenkins Rest API 8

How to interact with it 8

Why use it? 8

Security 8

Authentication versus authorization 8

Matrix security 8

Fingerprints 8

What are fingerprints? 8

How do fingerprints work? 8

Artifacts 9

How to use artifacts in Jenkins 9

Configuration Management (Tools such as Chef, Puppet, etc.) 9

Elements of software configuration management 9

Change management policies 9

Importance of software configuration management 9

Using 3rd party tools 9

How to use 3rd party tools with Jenkins 9

Chapter 2 – Jenkins Usage 10

Jobs 10

Organizing jobs in Jenkins 10

Parameterized jobs 10

Usage of Freestyle/Pipeline/Matrix/Maven/Literate 10

Builds 10

Setting up build steps and triggers 10

Configuring build tools 10

Running scripts as part of build steps 10

Source Code Management 11

Polling source code management 11

Creating hooks 11

Including version control tags and version information 11

Testing 11

Testing for code coverage 11

Test reports in Jenkins 11

Displaying test results 11

Integrating with test automation tools 12

Breaking builds 12

Notifications 12

Setup and usage 12

Email notifications, instant messaging, build radiators 12

Alarming on notifications 12

Distributed Builds 12

Setting up and running builds in parallel 12

Setting up and using SSH slaves, JNLP slaves, cloud slaves 12

Monitoring nodes 12

Plugins 12

Setting up and using Plugin Manager 12

Finding and configuring required plugins 13

CI/CD 13

Using Pipeline (formerly known as Workflow) 13

Integrating automated deployment 13

Release management process 13

Pipeline stage behavior 13

Jenkins Rest API 13

Using REST API to trigger jobs remotely, access job status, create/delete jobs 13

Security 14

Setting up and using security realms 14

User database, project security, Matrix security 14

Setting up and using auditing 14

Setting up and using credentials 14

Fingerprints 15

Fingerprinting jobs shared or copied between jobs 15

Artifacts 15

Copying artifacts 15

Using artifacts in Jenkins 15

Artifact retention policy 15

Alerts 15

Making basic updates to jobs and build scripts 15

Troubleshooting specific problems from build and test failure alerts 15

Chapter 3 – Building Continuous Delivery Pipelines 16

Pipeline Concepts 16

Value stream mapping for CD pipelines 16

Why create a pipeline? 16

Gates within a CD pipeline 16

How to protect centralized pipelines when multiple groups use same tools 16

Definition of binary reuse, automated deployment, multiple environments 16

Elements of your ideal CI/CD pipeline – tools 16

Key concepts in building scripts (including security/password, environment information, etc.) 16

Upstreams and downstreams 16

Triggering jobs from other jobs 16

Setting up the Parameterized Trigger plugin 17

Upstream/downstream jobs 17

Triggering 17

Triggering Jenkins on code changes 17

Difference between push and pull 17

When to use push vs pull 17

Pipeline (formerly known as Workflow) 17

Benefits of Pipeline vs linked jobs 17

Functionalities offered by Pipeline 17

How to use Pipeline 18

Pipeline stage concurrency 18

Visualization 18

Options to visualize jobs’ relationships 18

When to use various options for visualizing jobs’ relationships 18

Information offered by a build pipeline view 18

How to set up build pipeline visualization 18

Folders 18

How to control access to items in Jenkins with folders 18

Referencing jobs in folders 18

Parameters 18

Setting up test automation in Jenkins against an uploaded executable 18

Passing parameters between jobs 19

Identifying parameters and how to use them: file parameter, string parameter 19

Jenkins CLI parameters 19

Promotions 19

Promotion of a job 19

Why promote jobs? 19

How to use the Promoted Builds plugin 19

CD Metrics 19

KPIs/metrics for CI/CD 19

Determining how many builds failed, succeeded 19

Determining how long a build takes 20

Determining how often code is checked-in 20

How to use metrics/KPIs 20

Notifications 20

How to radiate information on CD pipelines to teams 20

Chapter 4 – CD as Code Best Practices 21

Distributed builds architecture 21

Fungible (replaceable) slaves 21

Master-slave connectors and protocol 21

Tool installations on slaves 21

Cloud slaves 21

Containerization 21

Traceability 21

High availability 21

Automatic repository builds 21

Chapter 5 – Cloudbees Jenkins Platform 22

Reference architecture 22

Role-based Access Control (RBAC) 22

Folders Plus 22

Templates 23

Setting up High Availability (HA) 23

CloudBees Jenkins Operations Center (CJOC) 23

Shared clouds 23

Cloud configurations 24

Shared slaves 24

Analytics 24

Cluster Operations 24

Pipeline Checkpoints (formerly known as Pipeline Checkpoints) 25

Custom Update Center 25

Multi-branch 26

Docker plugins 26

# Chapter 1 – Key CI/CD/Jenkins Concepts

## Continuous Delivery/Continuous Integration Concepts

### Define continuous integration, continuous delivery, continuous deployment

* Continuous integration – everyone commits to the mainline at least daily and automated build to verify
* Continuous delivery – can release to prod at any time via a push button deployment
* Continuous deployment – actually deploying to production continually
* DevOps is cultural and is broader than continuous delivery
* Pipeline has visibility, feedback and continuous delivery
* CI practices
  + Single source repository
  + Automate the build
  + Make your build self testing
  + Everyone commits everyday
  + Every commit triggers a build
  + Fix broken builds immediately
  + Keep the commit build fast (and use pipeline for slower builds)
  + Test in a clone of the prod environment
  + Make it easy to get the latest build
  + Visibility
  + Automate deployment
* CD principles
  + Check in
  + Build and unit tests
  + Automated acceptance tests
  + User acceptance test
  + Release

### Difference between CI and CD

* CI doesn’t require deploying

### Stages of CI and CD

* Start with the commit stage which compiles and runs unit tests.
* Then run longer tests/quality tools/ Ex: acceptance tests (given/when/then)
* Finally, deploy

### Continuous delivery versus continuous deployment

* Delivery means the ability to deploy to production. Deployment means actually doing so

## Jobs

### What are jobs in Jenkins?

* Job/Project – Runnable tasks

### Types of jobs

* Freestyle project
* Maven project
* Pipeline
* Multi configuration
* Multi branch
* Long running

### Scope of jobs

* *Not sure what this means – Maybe that there is a long running job type?*

## Builds

### What are builds in Jenkins?

* Build – Result of one run of a job/project

### What are build steps, triggers, artifacts, and repositories?

* Build step – a single operation withing a build
* Triggers – something that starts a build (time, SCM polling, etc)
* Artifact – output of a build
* Repository – the SCM system where the code to be built lives

### Build tools configuration

* In Manage System, set up location of tools like the JDK, Ant and Maven

## Source Code Management

### What are source code management systems and how are they used?

* Use to track code
* Client/server – one source of truth such as SVN.
* Distributed version control – every developer has copy of repository, peer to peer, such as Git.

### Cloud-based SCMs

* Ex: Git hub

### Jenkins changelogs

* List commits since last build

### Incremental updates v clean check out

* Incremental updates – faster
* Clean check out – guarantees no extra or changed local files

### Checking in code

* Should be at least daily with CI

### Infrastructure-as-Code

* Storing everything needed to build your environment

### Branch and Merge Strategies

* Branch by release
* Branch by feature – by user story
* Branch by abstraction – one branch, but turn features on/off by release
* Merge regularly

## Testing

### Benefits of testing with Jenkins

* Fast feedback!

### Define unit test, smoke test, acceptance test, automated verification/functional tests

* Unit test – test one class, often involves test doubles
* Integration/functional test – test components together
* Smoke test – sanity check to reject a release. Looking for major errors.
* Acceptance test – user level test for feature

## Notifications

### Types of notifications in Jenkins

* Failure, second failure, success, etc
* Active/push – radiators/SMS vs passive/pull – rss/dashboard
* RSS - /rssAll, /rssFailed and rssLatest
* Radiator view plugin uses the entire screen
* Extreme feedback – physical/audio devices

### Importance of notifications

* Fixing a build is high priority so need to know it is broken
* Communicating the status to all parties

## Distributed Builds

### What are distributed builds?

* Running builds on a different machine than master

### Functions of masters and slaves

* Master – basic Jenkins install
* Slaves – just for running jobs

## Plugins

### What are plugins?

* Add functionality to core Jenkins

### What is the plugin manager?

* UI for uploading/managing plugins

## Jenkins Rest API

### How to interact with it

* Format: XML or JSON
* Python and Ruby wrapper APIs

### Why use it?

* Programmatic access

## Security

### Authentication versus authorization

* Authentication – identify a user
* Authorization – what user can do

### Matrix security

* Maps roles to permissions
* Major categories: overall, slave, job, run, view and SCM

Definition of auditing, credentials, and other key security concepts

* Auditing – logging user operations and changes
* Credentials – username/password or the like for access

## Fingerprints

### What are fingerprints?

* MD5 checksum of files
* UI says for jar files, but works for any type of file

### How do fingerprints work?

* The first time you run a job with a post build step to generate a fingerprint, a new left navigation option shows up to check a file’s fingerprint.
* You can upload a file you have to see if any file Jenkins knows the fingerprint of matches.

## Artifacts

### How to use artifacts in Jenkins

* Download, put in Nexus, deploy, etc

Storing artifacts

* Can archive
* Can control discard policy

## Configuration Management (Tools such as Chef, Puppet, etc.)

### Elements of software configuration management

* Tracking/controlling changes in the software
* Includes version control

### Change management policies

* *Not sure what they mean here. This is a big topic*

### Importance of software configuration management

* Need to know what you deploy!

## Using 3rd party tools

### How to use 3rd party tools with Jenkins

* Setup in Manage System the location on disk or download from there
* Ex: JDK, Maven, Git
* Can install automatically or from file system

# Chapter 2 – Jenkins Usage

## Jobs

### Organizing jobs in Jenkins

* Jobs are organized in folders

### Parameterized jobs

* Check “This build is parameterized” and enter parameters/default values
* Run directly with “Build with Parameters” or call from upstream job with “trigger parameterized build” post build action and passing parameters

### Usage of Freestyle/Pipeline/Matrix/Maven/Literate

* Freestyle – most flexible job
* Pipeline – enter code in DSL. There is a snippet generator which generates the Groovy for common operations and lists the available environment variables.
* Matrix (multi-config) – Specify a configuration matrix with one or more dimensions. Runs all combinations when build.
  + Axis: slave, label (for slave) or user defined (string)
  + Combination filter: if don’t want cross product of all axis to run
  + Can execution “touchstone” builds first to specify which job(s) should run first and if this should skip the others
* Maven - less options than Freestyle since can assume based on Maven conventions
* Literate – brand new plugin (Dec 2015) – allows specifying build commands in README.md file in source control. A literate job is a type of multi-branch job. (searches for new branches and creates jobs in folder automatically)

## Builds

### Setting up build steps and triggers

* Common build steps include Maven/Ant, execute shell, start/stop Docker container
* Common triggers include time/periodic, SCM polling, upon completion of another job

### Configuring build tools

* In Manage Jenkins > Manage System
* Install automatically or via system

### Running scripts as part of build steps

* Can run OS script or Groovy script
* Groovy scripts can run as system or user level. System has access to Jenkins object model

## Source Code Management

### Polling source code management

* Set schedule using cron format
  + minute hour dayOfMonth month dayOfWeek
  + For dayofWeek, 0 is Sunday and 7 is Saturday
  + Can use H (or H/2 etc) for minute column to use a hash based on the job name to distribute jobs so don’t all start at the top of the hour.
  + Also support, @yearly, @annually, @monthly, @weekly, @daily, @hourly and @midnight
  + @Midnight means between midnight and one am since uses hash to distribute
* Required URL
* Optional credentials
* Options vary by repo. Ex: SVN lets you specify infinity/immediates/etc as checkout depth. Git lets you specify a branch specifier

### Creating hooks

* Hook script in repository triggers job
* Ex: Github plugin provides hook

### Including version control tags and version information

* Git allows you to create a tag for every build
* Version Number plugin lets you include info in build name

## Testing

### Testing for code coverage

* In build, must create XML file with data
* Post Build Action to publish
* For Java: Cobertura and JaCoCo
* In Cobertura, can set thresholds for weather icons:
  + Sunny - % higher than threshold
  + Stormy - % lower than threshold
  + Unstable - % lower than threshold
* In Jacoco, can set thresholds for sunny and stormy

### Test reports in Jenkins

* Publish JUnit or TestNG reports
* In JUnit, can set amplification factor - 1.0 means 10% failure rate scores 90% health. .1 means 10% failure rate scores 99% health.

### Displaying test results

* Configure as Post Build Action
* Point to xml files: ex: reports/\*.xml
* Can drill down to see details of tests runs and durations

### Integrating with test automation tools

* Can run acceptance tests later in pipeline than unit/component tests

### Breaking builds

* JUnit allows choosing whether to fail builds on test failures - default is “unstable” not failure

## Notifications

### Setup and usage

* Setup in post build action section

### Email notifications, instant messaging, build radiators

* Email
  + Same recipient for each one (except can add committers since passed)
* Email ext
  + lets you customize the message and tailor the recipients per trigger
  + can send on failing, still failing, unstable, still unstable, successful, etc
* Jabber and IRC for instant messaging
* Since build radiators are full screen, the only way to edit is to add /configure to the URL

### Alarming on notifications

* Extreme notifications can have a video or audio cue in the real world

## Distributed Builds

### Setting up and running builds in parallel

* Builds run on different executors
* Multi-configuration jobs run the pieces in parallel

### Setting up and using SSH slaves, JNLP slaves, cloud slaves

* Can launch local slaves with SSH (blocking or non-blocking IO), Java Web Start, command line on master or Windows service

### Monitoring nodes

* Monitoring page uses JMelody
* Memory/CPU/etc stats
* Can see heap dump/GC/etc

## Plugins

### Setting up and using Plugin Manager

* Can provide a HTTP proxy if needed
* Can specify alternate update center URL for JSON
* Listed installed plugins
* Can install/upgrade/uninstall plugin
* Can unpin plugin so doesn’t use specific version of plugin

### Finding and configuring required plugins

* Updates tab – for upgrading plugin already have
* Available tab – for downloading new plugins
* Advanced tab – for uploading plugin hpi/jpi file from disk
* Configure plugins on Manage Jenkins -> Manage System

## CI/CD

### Using Pipeline (formerly known as Workflow)

* Use DSL to specify jobs to be built
* Example: node { stage ‘x’ echo ‘1’ stage ‘y’ echo ‘2’ }
* Sample commands:
  + build 'jeanne-test'
  + svn - checkout
  + retry – retry body up to X times
  + timeout – limit time spent in block
  + stash/unstash
  + load – include a Groovy script
  + parallel – specify two branches to run in parallel and whether to failFast
* When run build, see table with column and duration for each stage. Row is build #. Cell color coded for pass/fail. Can see log for each stage.

### Integrating automated deployment

* Have the pipeline itself triggered by SCM
* Then the pipeline triggers the commit job first followed by the rest of the jobs in the pipeline
* The docker variable can be used as a build step in the pipeline or to surround other lines

### Release management process

* *Not sure what this refers to. Gates/approvals?*

### Pipeline stage behavior

* Stages run one at a time unless specify parallel
* A subsequent stage only runs if the prior one was successful

## Jenkins Rest API

### Using REST API to trigger jobs remotely, access job status, create/delete jobs

* /api shows docs for the REST API at that level of the object model
* /api/xml, /api/json, /api/json?pretty=true, /api/python and /api/python?pretty-true
* Choose “trigger builds remotely” on build and set token to allow POST call.
  + Run build: POST to JENKINS\_URL/job/job-name/build?token=MY\_TOKEN
  + Run build with reason: POST to JENKINS\_URL/job/job-name/build?token=MY\_TOKEN&cause=xyz
  + Run Parameterized Build: POST to JENKINS\_URL/job/job-name/buildWithParameters?token=MY\_TOKEN&param=xyz
* Error handling:
  + If try to call /build for parameterized job, get a 400 error
  + If try to call with wrong token, get a 403 error
  + If don’t choose “trigger builds remotely”, it worked
* CSRF
  + Get token at JENKINS\_URL/crumbIssuer/api/xml
  + Pass .crumb as header with POST
* All job (at top level) latest status: JENKINS\_URL/api/xml
* Build numbers and urls for a job: JENKINS\_URL/job/jobName/api/xml
* Build result and details: JENKINS\_URL/job/jobName/buildNumber/api/xml
* Create job: POST to JENKINS\_URL/createItem?name=jobName and post config.xml
* Delete job: POST to JENKINS\_URL/job/jobName/doDelete
* Enable job: POST to JENKINS\_URL/job/jobName/enable
* Disable job: POST to JENKINS\_URL/job/jobName/disable

## Security

### Setting up and using security realms

* Choices include Servlet Container, Google SSO, OpenId, Jenkins user database, LDAP, UNIX group/user database, JCOC SSO

### User database, project security, Matrix security

* People link shows user list + committers
* Matrix based security – control privileges granularly using user ids/groups
* Project based matrix authorization security – Matrix based + set privileges on job configuration page as well
* Role based matrix authorization security – Manage Roles to control permissions by group. Adds groups/roles tabs to projects

### Setting up and using auditing

* Manage Jenkins > System Log – for logging
* Job Configuration History plugin – for job config
* Audit Trail plugin – for Jenkins config

### Setting up and using credentials

* Domain – URL, host etc
* Credentials – username/password, cert, etc
* Use by choosing from pull down in job

## Fingerprints

### Fingerprinting jobs shared or copied between jobs

* Used to determine if a dependency has changed
* See which projects use a dependency
* See where fingerprinted files came from

## Artifacts

### Copying artifacts

* Build step to copy artifacts from another build
* Can choose which ones want to include/exclude by pattern

### Using artifacts in Jenkins

* Can refer to artifacts after build
* Treated specially not just as part of workspace

### Artifact retention policy

* By default, kept same length of time as build log.
* Can keep less time to save disk space

## Alerts

### Making basic updates to jobs and build scripts

* *Not sure what they mean here*

### Troubleshooting specific problems from build and test failure alerts

* *Not sure what they mean here*

# Chapter 3 – Building Continuous Delivery Pipelines

## Pipeline Concepts

### Value stream mapping for CD pipelines

* Entire process from concept to cash for a product
* Includes non code aspects such as product discovery
* Shows were time goes in process and where waits/delays are
* CD pipeline is subset of value stream map

### Why create a pipeline?

* Automated manifestation of process for getting software from version control to users
* Allows for phases of increasing fitness

### Gates within a CD pipeline

* Provide a point for approval before continuing.

### How to protect centralized pipelines when multiple groups use same tools

* *Not sure what this means. Approvals? Security?*

### Definition of binary reuse, automated deployment, multiple environments

* Binary reuse – Use other components as packaged, artifacts that have passed success criteria
* Automated deployment – using the same script to deploy to every environment
* Multiple environments – resources/configuration needed to work: ex: test, QA, Prod

### Elements of your ideal CI/CD pipeline – tools

* Source control repository
* Binary repository
* Automated testing
* Capacity testing
* Deployment

### Key concepts in building scripts (including security/password, environment information, etc.)

* Credentials plugin for password
* Keep environment information in source control
* Different script for each stage in the pipeline

## Upstreams and downstreams

### Triggering jobs from other jobs

* Build other projects
  + Comma separated list of jobs
  + Can specify to trigger only on good builds, good builds + unstable builds and always (even on failure)
  + All jobs share same trigger
* Trigger parameterized build on other projects
  + Comma separated list of jobs
  + Can control based on success, unstable, failure only, aborted, etc
  + Can set up multiple triggers so each set has different rules on when to run
  + Parameter types include boolean, string, from a property file, current build parameters, etc
  + Can pass through information like slave or Git/SVN trigger info

### Setting up the Parameterized Trigger plugin

* Check “This build is parameterized” and setup parameters
* Can use Node to specify slave by name from select list or label to specify slave’s build label

### Upstream/downstream jobs

* If A depends on B, B is the upstream job

## Triggering

### Triggering Jenkins on code changes

* For a commit build

### Difference between push and pull

* Pull - Set up a SCM polling trigger
* Push – Set up a hook from the repository to trigger job

### When to use push vs pull

* Pull for when you don’t control the repository or polling is ok
* Push for when you need an immediate build or don’t want to waste resources on polling

## Pipeline (formerly known as Workflow)

### Benefits of Pipeline vs linked jobs

* Scripted – can code loops/conditionals
* Resilient – can survive Jenkins restarts
* Pausable – can get manual approval
* Efficient – can restart from checkpoints
* Visualized – can see in dashboard

### Functionalities offered by Pipeline

* Build steps, pauses, parallelization, deploy, stash/unstash, etc
* Can run on certain node with node(‘master’) {}
* Can prompt user with input ‘query’
* Can do anything Groovy can do
* Can create stages

### How to use Pipeline

* Put commands want to run inside node{}
* Use snippet generated or write groovy script
* Can store global libraries in git at git clone <Jenkins>/workflowLibs.git

### Pipeline stage concurrency

* Parallel lets you run stages at same time

## Visualization

### Options to visualize jobs’ relationships

* Build Pipeline view – shows upstream/downstream dependencies for one job
* A pipeline automatically creates a stage view – can click to see “Full Stage View”
* Delivery pipeline view – not on exam? – shows more details about stages

### When to use various options for visualizing jobs’ relationships

* Can restrict to only include successful builds

### Information offered by a build pipeline view

* Dependencies
* Status
* When run

### How to set up build pipeline visualization

* Create a new view
* Choose job to start from
* Can also include in a dashboard view so have more than one per page

## Folders

### How to control access to items in Jenkins with folders

* Role Based Access Control can control folder
* Can control level as current/child/grandchild

### Referencing jobs in folders

* <jenkinsHome>/job/folder/job/name

## Parameters

### Setting up test automation in Jenkins against an uploaded executable

* File parameter in parameterized job
* Prompted to upload it when running manually

### Passing parameters between jobs

* Can type parameters, use property file, etc

### Identifying parameters and how to use them: file parameter, string parameter

* String parameter referred to by variable name ${TEST}
* File parameter placed in the workspace in the parameter name

### Jenkins CLI parameters

* Download jar from <Jenkins>/jnlpJars/jenkins-cli.jar
* Run as java –jar Jenkins-cli.jar –s <jenkinsUrl> help
* Add –noKeyAuth if don’t want to use SSH key

## Promotions

### Promotion of a job

* Can run steps after a gate
* Ex: archive artifacts, deploy, etc

### Why promote jobs?

* Way of communicating a build is good

### How to use the Promoted Builds plugin

* Promote Builds plugins lets you specify actions that require approval
* Adds promotion status link when check “Promote builds when…”
* Approvals include manually, automatically, based on downstream/upstream builds
* Can run multiple build steps (or post build actions) to run after approval – retry-able independently. Like a separate build.
* See icon once approved or if steps after approval fail
* Can have multiple promotion processes

## CD Metrics

### KPIs/metrics for CI/CD

* Cycle time
* Test coverage, cyclomatic complexity, duplication, etc
* Number of defects
* Velocity
* # Commits per day
* # Builds per day – success, failures and total
* Duration of build

### Determining how many builds failed, succeeded

* Dashboard view – build stats, job stats

### Determining how long a build takes

* Trend on individual job

### Determining how often code is checked-in

* Number of commit stage builds

### How to use metrics/KPIs

* Tracking improvement
* Identifying limiting constraint

## Notifications

### How to radiate information on CD pipelines to teams

* Email , radiator, etc

# Chapter 4 – CD as Code Best Practices

## Distributed builds architecture

* Run jobs on slave
* More secure because jobs run on slave
* More scalable because can add slaves
* Vertical growth – master is responsible for more jobs
* Horizontal growth – creation of more masters
* Recommend to virtualize slaves, but not master for performance

## Fungible (replaceable) slaves

* Can configure third party tools to automatically install on slaves
* Best practice is to make slaves interchangeable, but can tie jobs to slaves

## Master-slave connectors and protocol

* SSH connector – preferred option. Slaves need SSHD server and public/private key
* JNLP/TCP connector – Java Network Launch Protocol start web agent on slave through JWS (Java Web Start). Can start via browser or OS service
* JNLP/HTTP connector – like JNLP/TCP except headless and over HTTP
* Custom script – launch via command line

## Tool installations on slaves

* Can install manually or have Jenkins do it

## Cloud slaves

* EC2 for Amazon Cloud
* JCloud – for other clouds

## Containerization

* Docker image to deploy/run application
* “Build inside a Docker Container” option

## Traceability

* Docker Traceability plugin uses fingerprints for images

## High availability

* Master must be on network attached storage device
* Don’t do builds on master or at least not with workspace under JENKINS\_HOME
* HAProxy serves as the reverse proxy

## Automatic repository builds

* *Not sure what this means. It does not exist in any documentation online except the PDF study guide.*

# Chapter 5 – Cloudbees Jenkins Platform

## Reference architecture

* Products
  + Jenkins Enterprise – open source Jenkins plus plugins for High Availability, RBAC, Update Center, folders, etc.
  + Cloudbees Jenkins Operations Center – dashboard, manage multiple masters
* No builds on CJOC or downstream masters
* Recommend hundreds, not thousands of jobs on each downstream master
* Faster recovery and less frequent failures
* Proxy fronts primary master and checks availability
* CJOC master is a master with CJOC installed
* CJOC master knows about all slaves. Like a cloud for slaves
* Can set up different update centers for different downstream masters

## Role-based Access Control (RBAC)

* Setup in manage security. Choose role based matrix authorization strategy (vs matrix based on project matrix based)
* Defaults to logged in users can do anything and anonymous users can do nothing
* Default groups – Administrators, Developers, Browsers
* Default roles – anonymous, authenticated, administer, develop, browse
* Roles > Manage – global matrix of role/permission mappings
* Two types of roles – system defined and user defined
* Can’t get rid of anonymous and authenticated roles
* Extended read permission – can view, but not edit config
* Support group definitions out of the box – Jenkins, jobs, Maven modules, slaves, views and folders
* To prevent folder role from propogating to children - Group icons– blue means pinned
* To prevent folder role from inheriting from parent – Roles > filter

## Folders Plus

Features over folders plugin:

* tie slaves to folders
* move jobs between folders
* health reports other than child with worst health (ex: average health, job status, enabled projects)
* set icons on folder other than default (ex: aggregate of status, built in icons or by URL)
* pass environment variables to all jobs in folder
* display jobs from subfolders on higher level view
* restrict what goes in folder

## Templates

* Types
  + Auxiliary template – nested attributes within another template
  + Builder/publisher template – locked down builder/publisher
  + Folder/job template – configure folder/job
* If define in folder, limited to that folder
* Transformation types
  + Jelly – has ${} and some control tags – like JSTL but different tags.
  + Groovy template transformation – like a JSP in Groovy. Remember to backslash $
  + Groovy template for Pipleine
* Variables instance, model, parent (Folder or Jenkins instance itself) and parentInstance (the folder template where the job template sits)
* When admin updates template, automatically approved. When non-admin updates template, checked against whitelist of approved code or added to “in process script approval” list for admin.
* Groovy sandbox – can whitelist method signatures first time used. Format method class.Name methodName argTypes (or static method). Admins use whitelist too when sandbox on.
* Creating with REST
  + POST to /instantiate
  + Or /createItem and specify JobPropertyImpl for template

## Setting up High Availability (HA)

* HA for Jenkins is multiple JVMs forming a cluster.
* It is a singleon – only one is master at a time
* Config – NFS /shared disk, at least two servers, floating IP
* Jenkins-ha-monitor provides monitoring on when to switch IP between servers
* Need three pieces:
  + Jenkins enterprise war
  + Jenkins enterprise proxy HA war – start this and it proxies/passes through to regular Jenkins.war
  + Jenkins enterprise HA monitoring tool – triggers transfer logic from outside Jenkins
* Data survives failover except builds in progress and user sessions
* Typically takes a few minutes because has to start up secondary

## CloudBees Jenkins Operations Center (CJOC)

### Shared clouds

* Same access logic as shared slaves
* Clouds provision slaves to master
* Local Types: java web start or virtual machine

### Cloud configurations

* Supports Docker, Amazon EC2 and Microsoft Azure clouds
* Instance caps are managed on each master
* Credentials shared across masters

### Shared slaves

* Client masters in the same CJOC can share slave executors
* Client masters must be siblings or in same subfolder
* Slaves are leased to client masters for one job if CJOC is available. If it goes down, client master keeps slave until comes back.
* Client masters prefer slaves in current “folder” then go to parent
* Client masters are not allowed to use slaves at sibling folder level
* Create shared slaves with CJOC

## Analytics

* Jenkins masters report data to CJOC
* Display dashboards
* Can create custom dashboards
* To reindex and get historical data in CJOC
  + new Cluster Operations job
  + operation = masters
  + target masters == from operations root
  + step == reindex
* Can run Elastic Search embedded or remote
* Uses Kibana open source analytics and visualization platform
* Includes System/JVM metrics, Web UI metrics, Jenkins metrics, health checks
* Retention of data (reindexing resets clock)
  + Every 10 seconds metrics – saved 3 days
  + Hourly metrics – saved 3 years
  + Build reporting – saved 3 years
  + Other info saved forever

## Cluster Operations

* Used to performance maintenance operations from CJOC
* Ways to run
  + Checkbox on list view to prepare for shutdown or safe restart with left navigation “cluster operations”
  + Left navigation “cluster operations” on single master
  + Cluster operations job
* Each operation in job has:
  + type = master or update center
  + source = root, parent, parameter, etc
  + optional filter on path, online status, etc
  + steps
    - for master – Backup master, install/enable/disable plugin, execute groovy script, prepare for shutdown, refresh update center metadata, restart now, safe restart, upgrade jenkins, upgrade all plugins
    - for update center – Delete/promote/update core, delete/promote/update plugin, pull everything, pull new versions, refresh upstream sources, track latest core, track latest plugins
  + advanced options
    - # parallel items
    - timeout per step
    - failure mode – immediately, tidy (at end of current step), at end
    - build result to use on failure - unstable, failure, aborted
* If you have multiple items to operate on, they will occur in parallel

## Pipeline Checkpoints (formerly known as Pipeline Checkpoints)

* All pipelines can be resumed
* For a more granular resume, put checkpoint ‘name’ in your script.
* Local variables saved at checkpoint. Call stash if want to store files.
* Restart using Checkpoints link or retry icon
* Call unstash to retrieve files into workspace
* Get new build #, but skips all steps prior to checkpoint
* Place checkpoint outside of node{} so not reliant on state of workspace

## Custom Update Center

* Benefits: restricting plugins, sharing in house developed plugins
* Options:
  + Versions of plugins - Require explicit configuration or Implicitly push latest
  + Signature provider – ex: self signed
  + Upstream sources – like proxied updated centers – Jenkins Enterprise, Open Source or Local. Can also choose types: ex LTS
  + Maintenance tasks – pull new versions (of what already in update center) or pull everything
* Tabs
  + Core – Jenkins itself
  + Plugins – Jenkins plugins
  + Tool installers – ex: Groovy, Chrome Driver
  + Upload core – upload Jenkins war from local machine
  + Upload plugin – upload plugins from local machine
* Click Store button to save a version locally

## Multi-branch

* Benefits of Workflow Multi-Branch: automatic creation/deletion of job for each new/deleted branch in repo

and configuring properties by branch

* Uses marker file Jenkinsfile to define pipeline logic and recognize a job should be created
* Job gets deleted when branch or Jenkinsfile removed
* Create new Multibranch Workflow job
* Can give named branches different properties by specifying exceptions
* Creates a folder for these jobs to exist in

## Docker plugins

* Docker is containers for deployment
* Dockerhub (hub.docker.com) is like github – hosting for Docker
* Plugins
  + Docker – provision slave, run single build and then tear down that slave
  + Dockerhub notification - provides a hook so Docker can trigger Jenkins jobs when the image is updated
  + Docker build and registry - allows publishing to the Docker registry
  + Docker traceability – history of deployments/images
  + Docker pipeline provides docker variable to pipeline plugin
* Examples:
  + Build container: docker.build ‘path/app:${env.BUILD\_TAG}’
  + Run inside container: docker.image(‘name’).inside { /\* commands \*/ }
  + Reference container from outside in docker.withRun(‘name’).inside { /\* commands \*/ }