

# Table of Contents

---

1. [Introduction](#)
2. [Concepts In Go](#)
  - i. [Concepts in Go](#)
3. [Go Tour](#)
  - i. [Pipeline Dashboard](#)
  - ii. [Agents](#)
  - iii. [Agent Details](#)
  - iv. [Pipeline Activity](#)
  - v. [Stage Details](#)
  - vi. [Job Details](#)
  - vii. [Administration](#)
  - viii. [Server Details](#)
  - ix. [Environments](#)
  - x. [Value Stream Map](#)
4. [Installing Go](#)
  - i. [System requirements](#)
  - ii. [Installing Go Server](#)
  - iii. [Installing Go Agent](#)
  - iv. [Running Go without Installation](#)
  - v. [Upgrading Go](#)
  - vi. [Configuring Server Details](#)
  - vii. [Configure a Proxy](#)
  - viii. [Performance Tuning](#)
5. [Configuration](#)
  - i. [Setup a New Pipeline](#)
  - ii. [Managing Pipelines](#)
  - iii. [Managing Dependencies](#)
  - iv. [Managing Agents](#)
  - v. [Managing Environments](#)
  - vi. [Pipeline Labelling](#)
  - vii. [Pipeline Scheduling](#)
  - viii. [Parameterize a Pipeline](#)
  - ix. [Customize a Pipeline Label](#)
  - x. [Clone a Pipeline](#)
  - xi. [Lock a Pipeline](#)
  - xii. [Add Material to Existing Pipeline](#)

- xiii. [Add Stage to Existing Pipeline](#)
- xiv. [Add job to Existing Stage](#)
- xv. [Add task to Existing Job](#)
- xvi. [Pipeline Templates](#)
- xvii. [Choose When a Stage Runs](#)
- xviii. [Timer Trigger](#)
- xix. [Job Timeout](#)
- xx. [Managing Users](#)
- xxi. [Authentication](#)
- xxii. [Authorizing Users](#)
- xxiii. [Delegating Group Administration](#)
- xxiv. [Pipeline Group Administration](#)
- xxv. [Publish Reports and Artifacts](#)
- xxvi. [Managing Artifacts and Reports](#)
- xxvii. [Auto Delete Artifacts](#)
- xxviii. [UI Testing](#)
- xxix. [Mailhost Information](#)
- xxx. [Notifications](#)
- xxxi. [TFS Material configuration](#)
- xxxii. [Reference](#)
- xxxiii. [Schema](#)

## 6. Advanced Usage

- i. [Auto Register a Remote Agent](#)
- ii. [Spawn multiple instances of a Job](#)
- iii. [Multiple Agents on One Machine](#)
- iv. [Clean on Task Cancel](#)
- v. [Conditional Task Execution](#)
- vi. [Trigger With Options](#)
- vii. [Fan In](#)
- viii. [Properties](#)
- ix. [Compare Builds](#)
- x. [Graphs](#)
- xi. [Command Repository](#)
- xii. [Backup Go Server](#)

## 7. Integrating Go With Other Tools

- i. [Integration with External Tools](#)
- ii. [Gadgets](#)
- iii. [Mingle Integration](#)
- iv. [Displaying Mingle gadgets in Go](#)
- v. [Mingle Card Activity Gadget](#)

## 8. Go Api

- i. [Introducing Go API](#)
- ii. [Pipeline Group API](#)
- iii. [Pipeline API](#)
- iv. [Stages API](#)
- v. [Job API](#)
- vi. [Agent API](#)
- vii. [Materials API](#)
- viii. [Configuration API](#)
- ix. [Artifacts API](#)
- x. [Users API](#)
- xi. [Backup API](#)
- xii. [Properties API](#)
- xiii. [Feeds API](#)
- xiv. [Command Repo API](#)

## 9. Extension Points Of Go

- i. [Plugin User Guide](#)
- ii. [Package Repository Extension](#)
- iii. [Yum Repository Poller](#)
- iv. [Task Extension](#)

## 10. FAQ/Troubleshooting

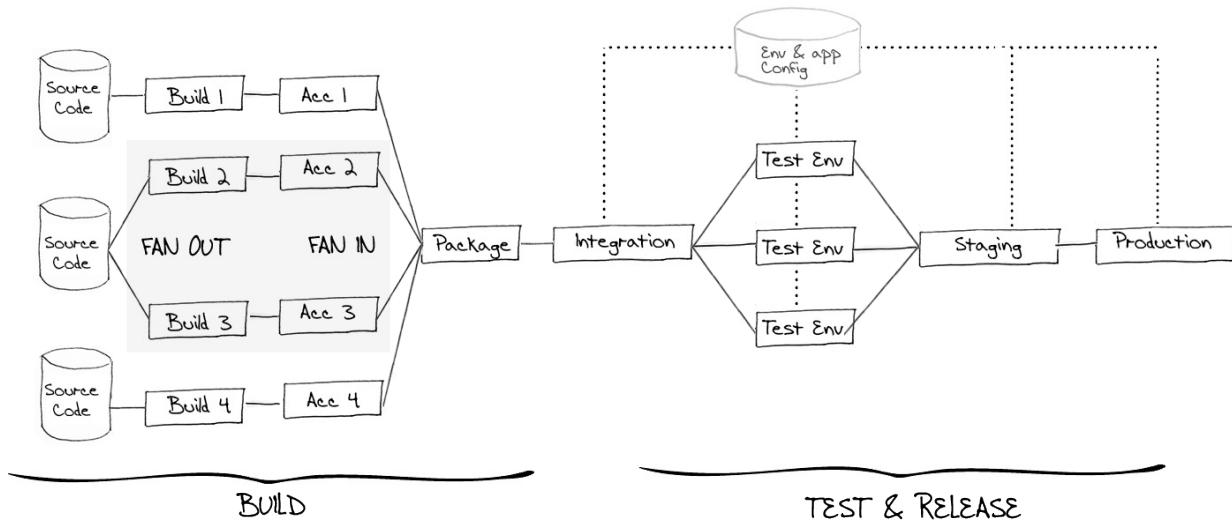
- i. [Ordering of Pipelines](#)
- ii. [Historical Configuration](#)
- iii. [Concurrent Modifications to Config](#)
- iv. [Why the Build is Broken?](#)
- v. [See artifacts as sub-tabs](#)
- vi. [Save Properties for a Build](#)
- vii. [Using Environment variables](#)
- viii. [Deploy to an environment](#)
- ix. [See changes in new binary](#)
- x. [Run Tests against new Builds](#)
- xi. [Check What's Deployed](#)
- xii. [Deploy a Specific Build](#)
- xiii. [Clone/Copy an Existing Agents](#)
- xiv. [OAuth Overview](#)
- xv. [What is OAuth?](#)
- xvi. [What is OpenSocial?](#)
- xvii. [How do I re-run jobs?](#)
- xviii. [Go unable to poll for changes](#)
- xix. [Artifact integrity verification](#)

- xx. [Email Notifications](#)
- xxi. [Running out of Disk Space](#)

- 11. [Beta features](#)
  - i. [Comment on a pipeline run](#)
- 12. [Release History](#)
  - i. [What's New in Go](#)

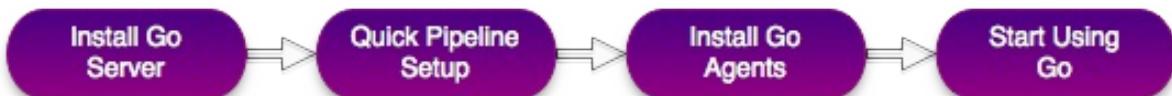
# Go User Documentation

Welcome to Go - an Open Sourced Continuous Integration and Release Management system. Please take a minute to understand [the basic concepts in Go](#) before you go any further.



## Quick help

Need help fast? Try out the topics below to get building in no time.



## Installation

- [System requirements](#) - Find out system requirements before you install Go.
- [Installing Go](#) - Learn how to install Go.
- [Upgrading Go](#) - Upgrade Go from an older version.

## Getting started

Installation done but need help setting up Go?

- [Set up your first pipeline](#) - Go captures the build plan for your project in a pipeline.

Find out how to set up your first pipeline.

- [Install agents](#) - You need a build agent in order to start building. Learn to deploy one or more agents here.

## Using Go

- [Concepts in Go](#) - Review basic concepts in Go. Understand how Go sees the world.
- [Managing pipelines](#) - Release Management with Go.
- [Managing a build cloud](#) - How to set up and scale your build cloud.
- [Managing artifacts and reports](#) - Keep your test reports, binaries and other artifacts in Go's central repository.

## See Go in use

[Watch tutorials on some common Go concepts and tasks](#)

## Contributing to this documentation

---

The code for this help documentation is hosted [here](#). If you think something can be improvised and/or added to the documentation, please send a [pull request](#) for the same. You can read more about pull requests [here](#).

We're using [GitBook](#) for generating our HTML Docs from markdown files. You can read more about it [here](#).

## Licence

---

Copyright 2014 ThoughtWorks, Inc. You may copy, distribute, print, and display, all or portions of this documentation, as long as you do so without making changes to the text. We also kindly request that you include a link to the original. All rights not granted above are reserved.

## Index (from old documentation)

---

- [Introduction](#)
  - [What's new in Go](#)
  - [Concepts in Go](#)
- [Installing Go](#)

- [System requirements](#)
- [Installing Go Server](#)
- [Installing Go Agent](#)
- [Running Go without Installation](#)
- [Upgrading Go](#)
- [Configuring Server Details](#)
- [Configure a Proxy](#)
- [Performance Tuning](#)
- Using Go
  - [Setup a New Pipeline](#)
  - [Managing Pipelines](#)
  - [Managing Agents](#)
  - [Managing Artifacts and Reports](#)
  - [Managing Dependencies](#)
  - [Managing Environments](#)
  - [Managing Users](#)
  - [Notifications](#)
  - [Properties](#)
  - [Pipeline Labelling](#)
  - [Compare Builds](#)
  - [Integration with External Tools](#)
  - [Ordering of Pipelines](#)
  - [Pipeline Scheduling](#)
  - [Gadgets](#)
  - [Auto Delete Artifacts](#)
  - [Job Timeout](#)
  - [Graphs](#)
  - [Historical Configuration](#)
  - [Command Repository](#)
  - [Concurrent Modifications to Go's Configuration](#)
  - [Package Material](#)
  - [Plugin User Guide](#)
  - [Fan In](#)
- Go Tour
  - [Pipeline Dashboard](#)
  - [Agents](#)
  - [Pipeline Activity](#)
  - [Stage Details](#)
  - [Job Details](#)
  - [Administration](#)

- Server Details
- Environments
- Value Stream Map
- As a Developer, I want to...
  - .. watch what's currently building
  - .. trigger a pipeline with a different revision of material
  - .. be notified when I break the build
  - .. understand why the build is broken
  - .. see my artifacts as a sub-tab on the Job Details page
  - .. save properties about a build
  - .. clean up my environment when I cancel a task
  - .. only run a task when the build has failed
  - .. use the current revision in my build
- As a Tester, I want to...
  - .. release something into my UAT environment
  - .. know what has changed in my new binary
  - .. ensure appropriate tests are run against new builds
- As a Release Manager, I want to...
  - .. release something into production
  - .. know what's currently in production
  - .. deploy a specific build to production
  - .. manage my environments
- As a Go Administrator, I want to...
  - .. template my pipelines
  - .. parameterize my pipelines
  - .. install a new agent
  - .. auto register a remote agent
  - .. clone/copy existing agents
  - .. install multiple agents on one machine
  - .. view and filter agents
  - .. add a new pipeline
  - .. clone a pipeline
  - .. add a new material to an existing pipeline
  - .. add a new stage to an existing pipeline
  - .. add a new job to an existing stage
  - .. add a task to an existing job
  - .. run the same job on a group of agents
  - .. pass environment variables to jobs
  - .. ensure only one instance of a pipeline can run at the same time
  - .. choose when a certain stage runs

- .. use a custom pipeline label
  - .. manage my dependent pipelines
  - .. enable authentication on my Go server
  - .. configure LDAP access for my Go server
  - .. change permissions for different actions
  - .. ensure only certain users can see a group of pipelines
  - .. publish reports and artifacts
  - .. configure an agent to run UI tests
  - .. add mailhost information to support email notifications
  - .. clean up old artifacts when running out of disk space
  - .. run a pipeline on a schedule
  - .. pause an agent
  - .. see if a job fails because of an environment issue
  - .. delegating group administration
  - .. backup Go server
  - .. be notified when Go server is not able to poll for changes
  - .. manage pipelines in my pipeline groups
- Mingle Integration
    - Displaying Mingle gadgets in Go
    - Mingle Card Activity Gadget
  - FAQ/Troubleshooting
    - What is OAuth?
    - What is OpenSocial?
    - How do I re-run jobs?
    - Go unable to poll for changes
    - Deploy specific revisions of the materials to an environment
    - Artifact integrity verification
    - Setting up a new agent by cloning an existing agent
    - How can I customise my email notifications?
  - Go API
    - Pipeline Group API
    - Pipeline API
    - Stages API
    - Job API
    - Agent API
    - Materials API
    - Configuration API
    - Artifacts API
    - Users API
    - Backup API

- [Properties API](#)
- [Feeds API](#)
- [Command Repo API](#)
- [OAuth Overview](#)
- Bundled Plugins
  - [Yum Repository Poller](#)
- Configuration
  - [Reference](#)
  - [Schema](#)
  - [TFS Material configuration](#)

# **CONCEPTS**

---

**IN GO**

---

# Introduction

---

Go is an advanced Continuous Integration and Release Management system. It takes an innovative approach to managing the build, test and release process. In order to find your way around Go, you'll need to understand how Go sees the world. This page explains some basic concepts in Go.

If you want to know more about Continuous Integration and delivery, in general, refer to Martin Fowler's article on the subject: [Continuous Integration](#) and [Continuous Delivery](#).

## Build Agents

---

As with all modern continuous integration systems, Go lets you distribute your builds across many computers -- think 'build grid' or 'build cloud'.

And why use a build cloud? There are three main reasons:

- Run your tests on several different platforms to make sure your software works on all of them
- Split your tests into several parallel suites and run them at the same time to get results faster
- Manage all your environments centrally so you can promote builds from one environment to the next

It is extremely simple to get agents up and running in Go. First, install the Go agent software on each computer that is to be a part of your cloud. Next, configure each build agent to connect to your Go server. Finally, approve every build agent in your cloud from the management dashboard in the Go administration page. You are now ready to build. Additionally, you should associate relevant **resource** tags with each of your agents to better specify the kinds of build tasks with which each agent is compatible.

## Agent lifecycle

Go supports a grid of Agents that can run Jobs. Agents periodically contact the Go server to ask if there is any work. The server checks to see what resources the Agent has, and assigns any available Jobs that match those resources.

When an Agent runs a Job, it creates a sandbox directory under the agent's directory. All

materials are updated into this directory. This directory may be different on different agents depending on how the agent was configured and which operating system the agent is running on.

For example consider a pipeline named "pipelines/my-pipeline". On a default linux install this would be "/var/lib/go-agent/pipelines/my-pipeline". On a default Windows installation it would be "C:\Program files\Go Agent\pipelines\my-pipeline".

All materials are updated in this directory. If a material has a "dest" folder specified, then the material is updated into a folder of that name, under the base sandbox directory.

For example suppose the pipeline has an SVN material with a destination folder name of "tools" then the svn files will be checked out into that "tools" directory. When you have multiple materials specified, you must specify a "dest" for each material.

Once the materials have all been updated, the Agent runs each of the tasks in turn. You can specify a workingdir on a task. When the task runs it will be run relative to the pipeline sandbox.

For example if there is a task that runs "/usr/bin/make" with a workingdir of "tools/my-tool" then this will run in the directory "pipelines/my-pipeline/tools/my-tool".

When all the tasks have been completed the agent will publish any artifacts defined in the Job. Again the artifact directories are relative to the pipeline sandbox directory.

## Full example

Consider the following configuration:

```
<pipeline name="my-product">
  <materials>
    <svn url="http://my-svn-server/tools" dest="tools"/>
    <svn url="http://my-svn-server/my-project" dest="my-project"/>
  </materials>
  <stage name="compile">
    <job name="linux">
      <environmentvariables>
        <variable name="FOO">
          <value>bar</value>
        </variable>
      </environmentvariables>
      <resources>
        <resource>linux</resource>
      </resources>
      <tasks>
        <exec command="/usr/bin/make" args="-f Makefile" workingdir="tools,>
```

```

<ant target="unit-test" workingdir="my-project"/>
</tasks>
<artifacts>
  <artifact src="my-project/target/deployable.jar" dest="pkg"/>
</artifacts>
</job>
</stage>
</pipeline>

```

The Go Agent will:

- Create the directory "[install-dir]/pipelines/my-product" if it does not exist
- Checkout or update the svn material "<http://my-svn-server/tools>" into "[install-dir]/pipelines/my-product/tools"
- Checkout or update the svn material "<http://my-svn-server/my-project>" into "[install-dir]/pipelines/my-product/my-project"
- run "/usr/bin/make" in the directory "[install-dir]/pipelines/my-product/tools/my-tool"
- run ant in the directory "[install-dir]/pipelines/my-product/tools/my-project"
- publish "[install-dir]/pipelines/my-product/my-project/target/deployable.jar" to the server

## Pipelines

---

A pipeline allows you to break down a complex build into a sequence of simple stages for fast feedback, exhaustive validation and continuous deployment.

### How Go models distributed work

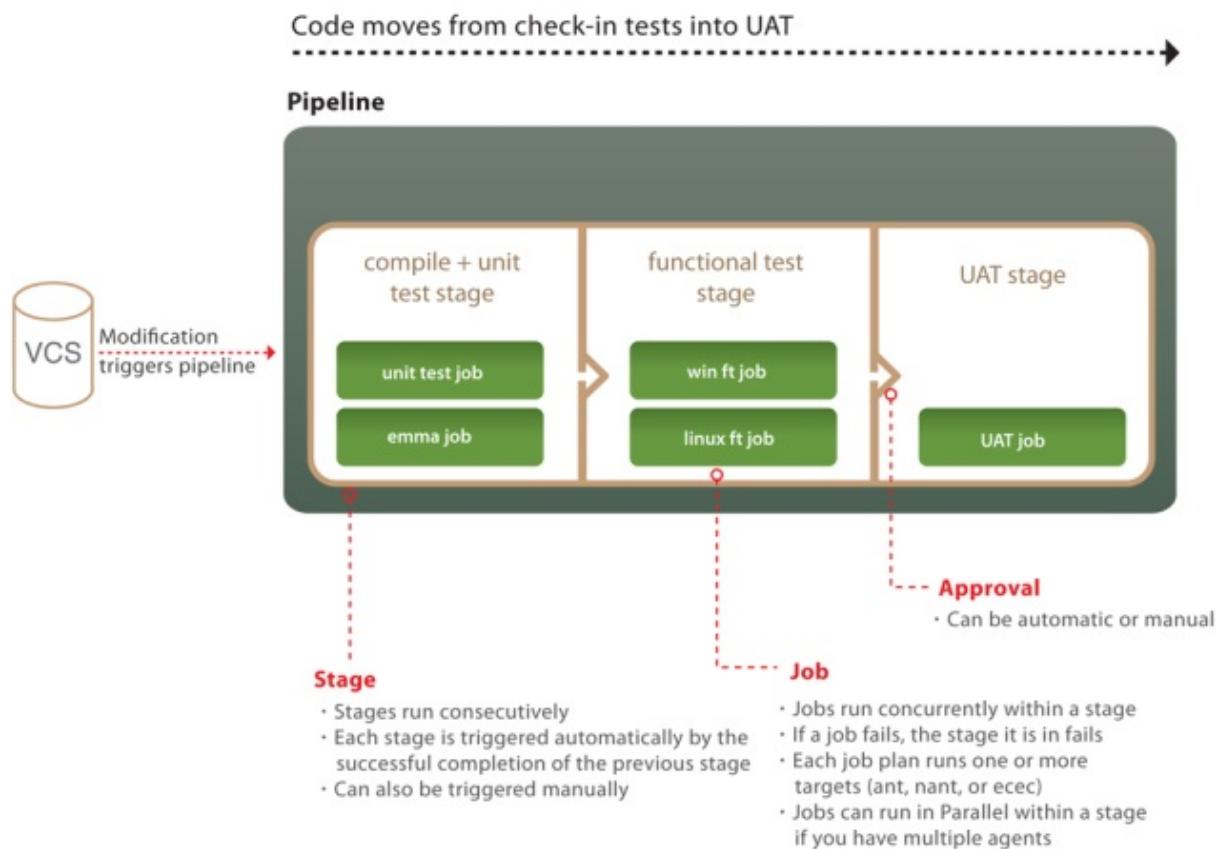
The unit of work in Go is called a **job**. A job is a set of build tasks that can be performed on a single agent in your cloud. You can associate specific build **resources** with each build agent -- a specific operating system or compiler version, for example. Go makes sure build jobs that require specific build resources are directed to build agents with the appropriate resources. By default, build jobs can be picked up by any agent. Resources are simple text tags which you associate with each agent. You can specify as many of them as you want. This flexibility is important as the agent process itself does not automatically determine anything about its environment.

Jobs are grouped into stages. A **stage** is a collection of build jobs that can be executed in parallel. This is the mechanism that allows you to, for example, split test suites into multiple parallel streams or run the same build on multiple platforms simultaneously. A stage passes only when all the jobs in the stage pass.

Stages are then joined sequentially into a **pipeline**. Stages trigger in the order they appear in the pipeline's raw configuration. They can be triggered by: a change in your version control system, manually forcing the pipeline to become active or by a dependency on a given stage of another pipeline. When a stage completes successfully, it triggers the next stage in the pipeline automatically, by default. Alternatively, you can require a manual **approval** to trigger the next stage. This manual approval requires user intervention. You can delegate the permissions for approval of stages to individuals or groups of users.

## An example pipeline

So what does a pipeline look like? Here's an example:



The first stage has two jobs. The unit test job compiles the code and runs the unit tests. The compiled code is then uploaded to the artifact repository. This is the one and only time the code is compiled -- and of course if you're using an interpreted language you can skip this step. The second job does static analysis of the code, uploading the results as html test reports and build properties for further analysis.

When the first stage passes, it automatically triggers the functional test stage. The jobs in this stage download that binaries from the artifact repository, and run a series of

functional tests. One job runs on a Linux box, the other on Windows. If your tests take a long time to run, you could split them into suites and run these as multiple jobs in parallel.

Finally there is a stage which deploys your software into your UAT environment for manual testing. This stage has a manual approval in front of it, meaning that somebody has to click a button in order to deploy the application into UAT. Running this stage proves out your automated deployment process -- and it should include some smoke tests that make the job fail if the deployment doesn't work.

The pipeline metaphor gives you several important benefits:

- Because of the way pipelines are modeled and presented, it is trivially easy to match up an acceptance test failure, or a flaw in the UAT environment, with the version of the code that caused it.
- Because you only compile once, you ensure that the thing you are testing is the same thing you will release, and you don't waste resources compiling repeatedly.
- Finally, Go allows you to build manual steps into your testing process so that your QAs and users can manually test your software.

## Pipeline groups

The Go dashboard allows you to visualize pipelines at a glance. You can then group pipelines together into pipeline groups. Beyond the visual convenience of grouping related pipelines, there are access and security features that allow you to control the set of users that can view a particular pipeline group. This can be a powerful way to carve out more secluded build areas for your users. Consult the Security section for more information

## Pipeline dependencies

---

A dependency is created when a pipeline is configured to depend on either an SCM or another pipeline. The domain concept for this is called a material. If a pipeline has a dependency it has been configured to have either an *SCM material* or a *pipeline material*.

### SCM Dependency



Pipeline 1 depends on an SCM.

Pipeline 1 will trigger each time it polls the SCM and detects a new revision.

## Pipeline Dependency



Pipeline 2 depends on Pipeline 1.

Pipeline 2 will trigger each time Pipeline 1 successfully completes.

## Pipeline and SCM Dependency



Pipeline 2 depends on Pipeline 1 and an SCM.

The expectation may be that a new revision will trigger both Pipeline 1 and 2. That's not the case.

Instead, Pipeline 2 will only trigger if a new revision has successfully passed through Pipeline 1.

Real world example:

SCM contains application code and acceptance tests

Both pipelines depend on the SCM

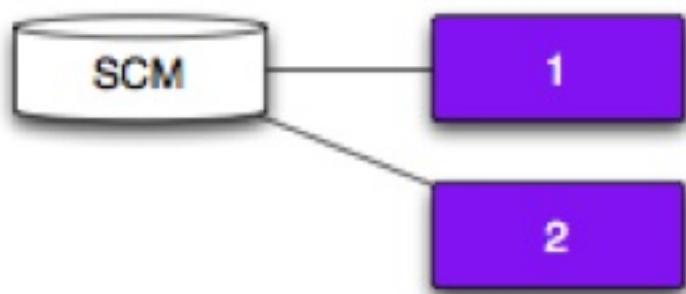
Pipeline 1 compiles code and creates an artifact

Pipeline 2 fetches the build artifact and runs the tests that were written for the compiled code

Pipeline 2 has to use the same acceptance tests committed with the code

Someone commits code and tests as part of the same SCM commit.

## Fan-out



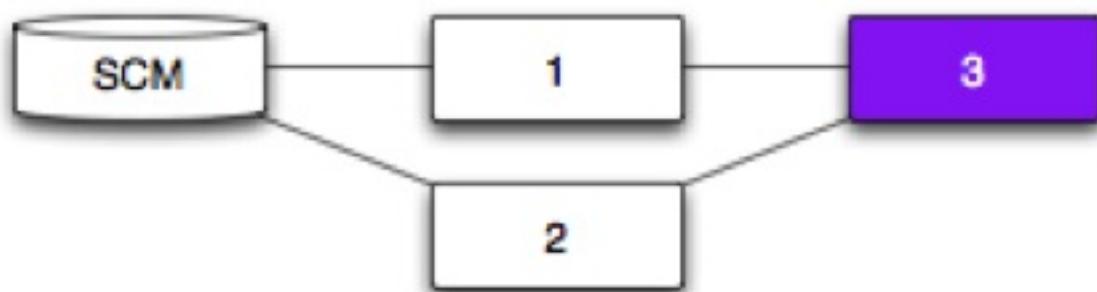
Pipeline 1 and 2 depend on the same SCM.

A new revision will trigger both Pipeline 1 and 2.

Why fan-out?

Speed up the feedback loop and enable a team to fail and learn faster.

## Fan-in - Pipeline



Pipelines 1 and 2 depend on an SCM.

Pipeline 3 depends on Pipelines 1 and 2.

The expectation may be that any successful run of Pipeline 1 or Pipeline 2 will trigger Pipeline 3. That's not the case.

Instead, Pipeline 3 will only trigger if a new revision (SCM) has successfully passed through both Pipeline 1 and Pipeline 2.

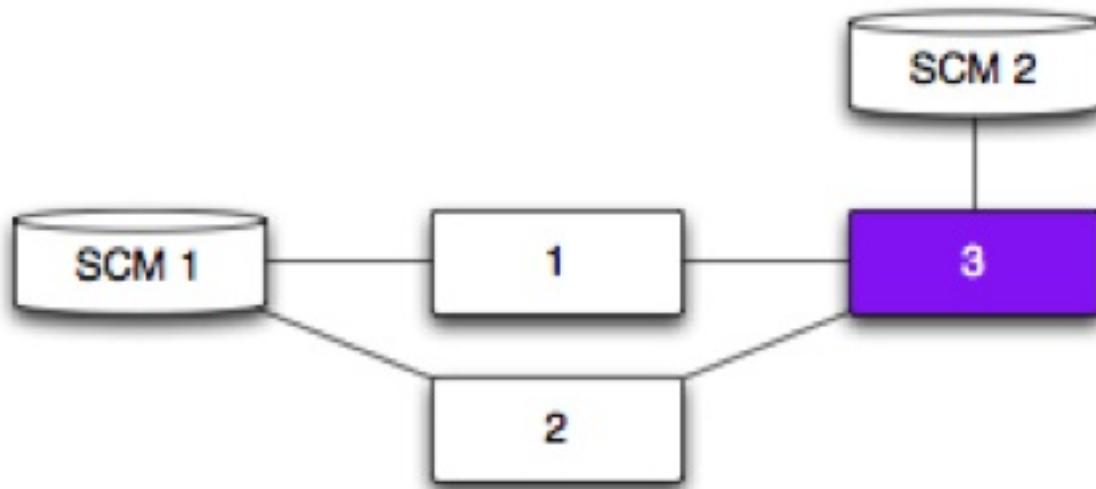
Why fan-in?

Faster feedback - You can fan-out to get significantly faster feedback now that Go handles the fan-in.

Eliminates spurious builds - Test software only when it's necessary and don't waste resources.

Meaningful feedback - You won't have false-positives or false-negatives.

## Fan-in - Pipeline and SCM



Pipelines 1 and 2 depend on SCM 1.

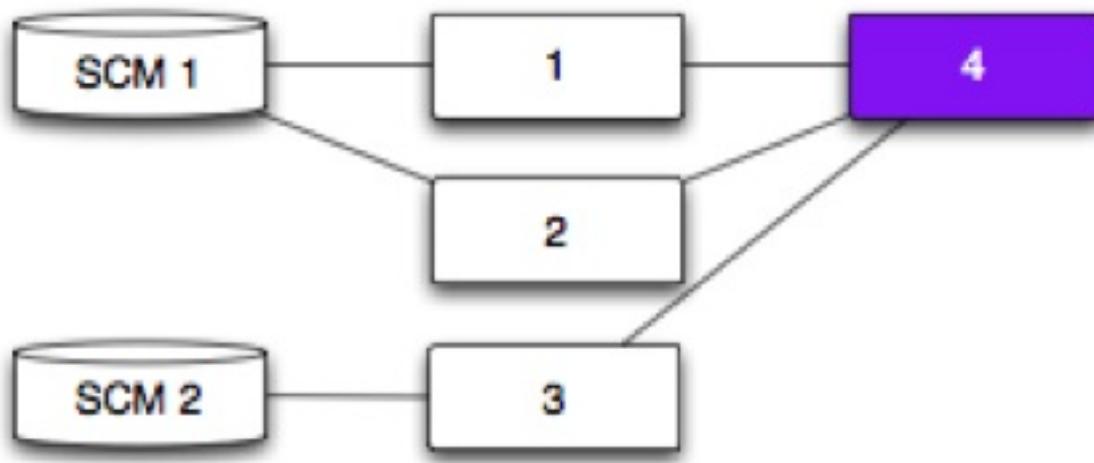
Pipeline 3 depends on Pipelines 1 and 2, and SCM 2.

Pipeline 3 will trigger:

if a new revision (SCM 1) has successfully passed through both Pipeline 1 and Pipeline 2

each time it polls SCM 2 and detects a new revision

## Fan-in - Pipeline with different originating SCM



Pipelines 1 and 2 depend on SCM 1.

Pipeline 3 depends on SCM 2.

Pipeline 4 depends on Pipelines 1, 2 and 3.

Pipeline 4 will trigger:

if a new revision (SCM 1) has successfully passed through Pipelines 1 and 2.

each time Pipeline 3 successfully completes (SCM 2)

## Deployment and Environments

---

Any moderately complex piece of software typically goes through a series of environments prior to release. A typical set of environments may look like this:

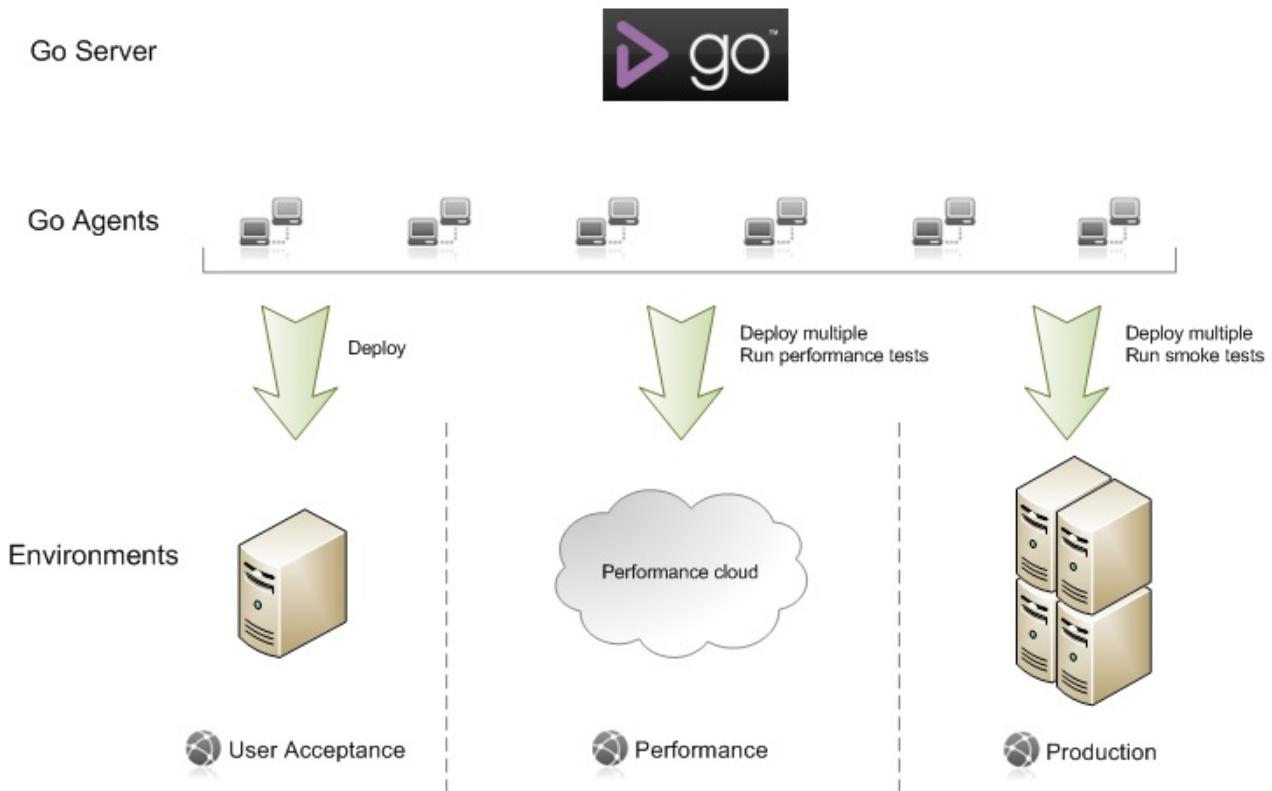
- Development
- User Acceptance Testing
- Performance
- Production

Deployment of software into an environment is often a non-trivial process that involves several steps, potentially required to be repeated on several machines. With Go, you have an easy to manage environment dashboard that simplifies the deployment process down to a single click.

An environment in Go is essentially a collection of agents and pipelines. Agents can be shared across environments, however, pipelines cannot. An environment pipeline is used to perform tasks specific to an environment.

## An example

So how would this work for me? Here's an example:



In this scenario, we have three different environments -- User acceptance, Performance, and Production. At any given time, each environment may contain a different build of the deployed software.

Each environment has a pipeline (represented by the downward arrow) that performs a set of tasks specific to each environment.

- **User Acceptance:** The pipeline performs a **deploy** task on a single machine.
- **Performance:** The pipeline first performs a **deploy** on a cluster of performance machines and then triggers a suite of **performance tests**.
- **Production:** The pipeline first performs a **deploy** on a load balanced cluster of machines and then triggers a suite of **smoke tests** to validate the deployment.

Each of these environment pipelines deploys using the same build thereby mitigating any unforeseen issues due to inconsistent versions.

The environments dashboard also give you a bird's eye view of which versions are deployed in which environment giving you a configuration management console as well.

**Go**

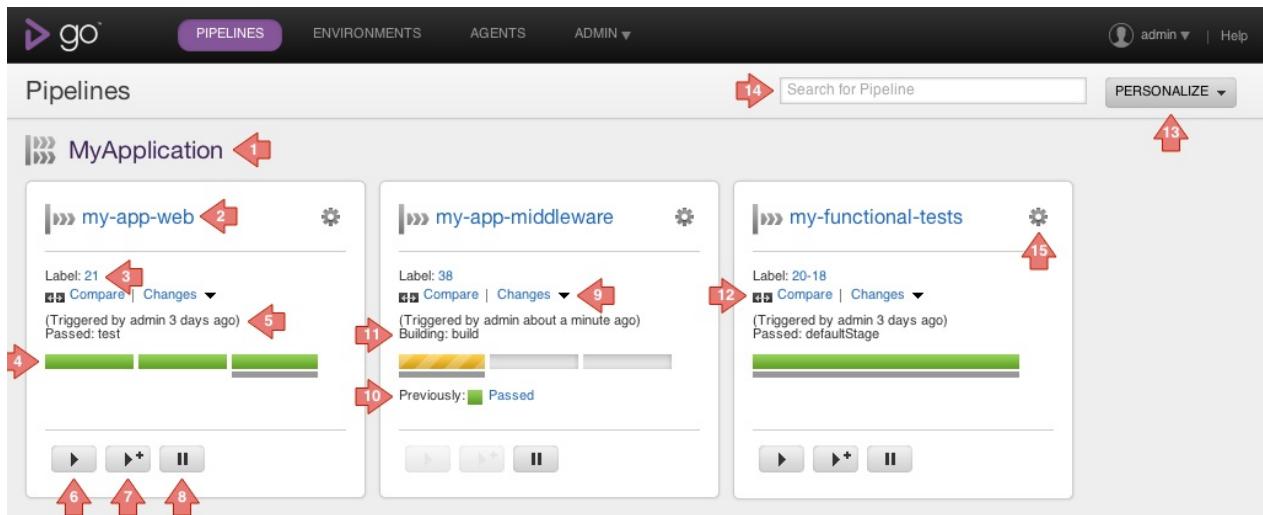
---

**TOUR**

---

# Pipelines Dashboard

This shows the current activity in the system. You can see all the instances of a given pipeline that are currently in progress.

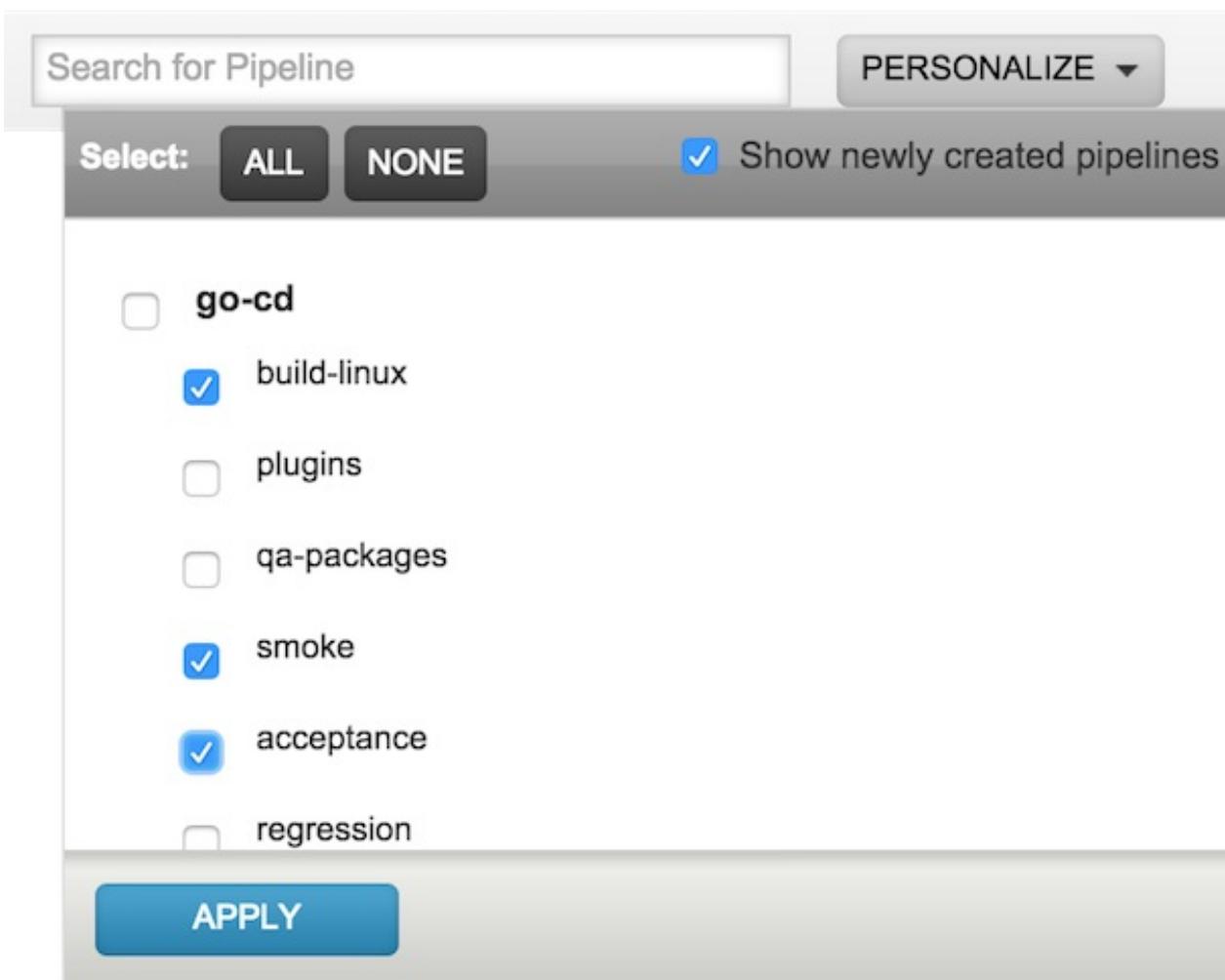


## Key

1. The pipelines are listed under the pipeline group that they belong to
2. Click the pipeline name to see [pipeline activity](#) for that pipeline.
3. Click pipeline instance label to see details for that instance.
4. Click on each stage segment to see [stage details](#) for that stage instance.
5. "triggered by [user name] about [how long ago]" gives you a quick look at who activated this pipeline and when this was triggered.
6. The "Trigger" button forces a pipeline to begin build activity
7. The [Trigger with Options](#) button allows the you to pick the revisions of materials that the pipelines should build with, and trigger the pipeline.
8. The "Pause" button pauses scheduling of the pipeline.
9. The "Changes" shows you the modifications to materials that have been built in this instance. The "!" indicates that the changes are being built for the first time.
10. "previously:[status]" tells you what the status of the currently running stage in the previous pipeline instance was. The previous instance is based on [natural ordering](#). On hover, you can see the label of the 'previous' pipeline instance.
11. This shows you the name and status of the last executed stage in that pipeline instance.
12. View all changes between the current pipeline instance with the previous one.
13. Lets you customize which pipelines are displayed on the dashboard. See [below](#) for more details.

14. Lets you search for any pipeline configured to be visible on your pipeline dashboard.
15. If you are a Go pipeline group administrator or a super administrator, you can now navigate to edit a pipeline by clicking this settings icon on the pipeline dashboard or the environments page.

## Personalize pipelines view



You can customize and control the pipelines you see on the dashboard by using the "Personalize" button on your dashboard. You can choose which pipelines you want to see, in this list, and save your selection by clicking on the "Apply" button. From that point onwards, the pipelines dashboard will only show your selections.

However, at a later point, if a new pipeline is added by someone else, that pipeline shows up on your dashboard as well. Uncheck the "Show newly created pipelines" checkbox to prevent them from showing up on your dashboard.

Pipelines created by you using the pipeline creation wizard will always be shown on your dashboard. You can remove them from your view after they are created.

**Also see...**

- Pipeline activity
- Job details
- Clean up after canceling a task
- Go overview

# Agents

The Agents page lists all the agents available to the server and their current status.

When an Agent first connects to the Server it is 'Pending'. An administrator must enable the Agent before Go will schedule work on that agent.

Administrators can also disable agents. Go will not schedule work for a disabled Agent. If a job is building on the agent when it is disabled, that job will be completed; the agent is then disabled. An administrator will need to enable the Agent before it will again schedule work

Administrators can choose to delete an agent which is no longer required. The agent must be disabled before it can be deleted. An agent in a disabled(building) or disabled(cancelled) state cannot be deleted.

The screenshot shows the Go Agents interface. At the top, there are navigation tabs: PIPELINES (4), ENVIRONMENTS (9), AGENTS (5), ADMIN (6), and Help (8). Below the tabs is a search bar with a placeholder 'tag: value' and buttons for FILTER (blue) and CLEAR (black).

Under the AGENTS tab, there are three buttons: ENABLE, DISABLE, and DELETE. There are also dropdown menus for RESOURCES and ENVIRONMENTS.

Below the buttons, a summary shows Pending: 0, Enabled: 4, and Disabled: 1 (1).

The main table lists agents with columns: Agent Name (checkbox), Sandbox, OS, IP Address, Status (sorted by default, 2), Free Space, Resources, and Environments.

Agent Name	Sandbox	OS	IP Address	Status	Free Space	Resources	Environments
blrstdcrspair03.thoughtworks.com /tmp/go-agent-2	Sandbox	Linux	10.4.7.203	building	233.6 GB	dev	no environments specified
blrstdcrspair03.thoughtworks.com /tmp/go-agent-4	Sandbox	Linux	10.4.7.203	building	233.6 GB	plugin	no environments specified
blrstdcrspair03.thoughtworks.com go-agent-5	Sandbox	Linux	10.4.7.203	building	233.6 GB	UAT01-twist-linux   UAT02-twist-linux	no environments specified
blrstdcrspair03.thoughtworks.com /tmp/go-agent-3	Sandbox	Linux	10.4.7.203	idle	233.6 GB	dist	no environments specified
blrstdcrspair03.thoughtworks.com /tmp/go-agent-1	Sandbox	Linux	10.4.7.203	disabled	233.6 GB	UAT01-twist-win   performance	no environments specified

A red arrow labeled 7 points to the 'go-agent-5' entry in the table.

## Key

1. Find out how many agents are pending, enabled and disabled.
2. Status is sorted by default. The order of sort is pending, lost contact, missing, building, idle, disabled, cancelled.
3. Click on a column header to sort by that column.
4. To enable or disable agents, first select the agents that you are interested in. Then click the 'ENABLE' or 'DISABLE' button. If you try to disable an agent that is already disabled, or enable an agent that is already enabled, Go will ignore that change.
5. To associate a resource with an agent, first select the agents you are interested in. Then click the 'Resources' button. You are now able to associate new or existing resources with your agents.
6. To associate an agent with an environment, first select the agents you are interested in. Then click the 'Environments' button. You are now able to associate

your agents with an environment.

7. Admin users can click here to get to the [Agent details](#) of the given agent
  8. Filter the agents list. See the section below.
  9. To delete agents, first select the agents that you are interested in. Then click the 'DELETE' button. If you try to delete an agent that is in disabled(building) or disabled(cancelled), Go will not delete that agent.

# Filtering Agents

Since the agent list can become very long, it is useful to be able to filter it by various criteria. The **Filter** option provides this functionality.

- Format: tag:value
  - Supported tags: ip, resource, os, name, status, environment
  - Supported values: Free form text. After you specify a tag, enter a value. Go only displays agents containing the entered value.
  - The sort function will work with filtered lists.
  - Entering a tag:value combination that does not match any agents will result in an empty result set being displayed.

## Examples:

- If you want to see only missing agents, enter "status:missing".
  - If you want to see only agents with resource names containing "java", enter "resource:java".

# Autocompletion

Go support autocomplete of searches. After you specify a tag, Go suggests possible values for the tag, based on the existing values entered. You can choose an appropriate value from the autocomplete list and then search.

Agents		ENABLE	DISABLE	DELETE	RESOURCES ▾	ENVIRONMENTS ▾	resource:U	FILTER	CLEAR
Pending: 0	Enabled: 4	Disabled: 1					UAT01-twist-win		
	Agent Name	Sandbox	OS	IP Address	Status	Free Space	UAT01-twist-linux	ts	
<input type="checkbox"/>	blrstdcrspair03.thoughtworks.com	/tmp/go-agent-5	Linux	10.4.7.203	lost contact	Unknown	UAT01-twist-linux   UAT02-twist-linux	no environments specified	
<input type="checkbox"/>	blrstdcrspair03.thoughtworks.com	/tmp/go-agent-2	Linux	10.4.7.203	idle	233.6 GB	dev	no environments specified	
<input type="checkbox"/>	blrstdcrspair03.thoughtworks.com	/tmp/go-agent-3	Linux	10.4.7.203	idle	233.6 GB	dist	no environments specified	
<input type="checkbox"/>	blrstdcrspair03.thoughtworks.com	/tmp/go-agent-4	Linux	10.4.7.203	idle	233.6 GB	plugin	no environments specified	
<input type="checkbox"/>	blrstdcrspair03.thoughtworks.com			10.4.7.203	disabled	Unknown	UAT01-twist-win   performance	no environments specified	

A maximum of 10 results is displayed for autocomplete, irrespective of the number of actual matches. For best results, ensure sufficient text is entered to narrow down the number of matches

## Exact search

Even with autocomplete, there are some limitations to the search criteria.

For example **resource:windows** matches both "windows" and "windows2k3", though you may have wanted an exact match on "windows". This is because, the current filter is a wildcard search rather than a token search.

**Exact search** addresses this problem. The way to do exact search is to use quotes ("") to specify the values. e.g. **resource:"windows"** will filter and return only those agents whose names are "windows" and nothing else.

You cannot combine autocomplete and exact search. Once you specify the values in quotes, autocomplete will be turned off.

## Also see...

- [Managing Agents](#)
- [Go overview](#)

# Details of a single agent

---

Go now provides a page that shows the details of a single agent. This page provides details about the agent configuration and the history of all the jobs that ran on that agent.

## Agent Details tab

---

This tab shows the configuration and runtime information of an agent. For example, this tab shows the free space available on the agent, the IP Address and the OS of the agent.

In terms of configuration, this tab shows the resources of the agent and the environment it belongs to. A sample Details tab looks as below:

The screenshot shows the Go application interface with the following details:

- Header:** go (with a purple arrow icon), PIPELINES, ENVIRONMENTS, AGENTS (highlighted in purple), ADMIN.
- Agent Name:** blrstdcrsato04
- Tab Bar:** Details (selected), Job Run History.
- Configuration Data:**

Free Space:	Unknown
Sandbox:	C:\AutoDeployTesting\uat-agent
IP Address:	10.4.8.84
OS:	Windows 2003
Resources:	Windows2003   autodeploy   windows
Environments:	AutoDeploy-WindowsServer2003   sriram-spike-windows

## Job Run History tab

---

You must be logged in as an admin user to configure this step.

This tab shows a table of all the completed jobs that ran on this agent. A sample page is shown below

Details	Job Run History			
Pipeline	Stage	Job	Result	Completed
sriram-windows-remote-cbgd	upgrade-go-server	upgrade	Passed	2013-02-11T10:36:42+05:30
sriram-windows-remote-cbgd	transferFile2node	robocopy	Passed	2013-02-11T10:33:01+05:30
sriram-windows-remote-cbgd	transferFile	fetch-go-binaries	Passed	2013-02-11T10:31:37+05:30
sriram-windows-remote-cbgd	upgrade-go-server	upgrade	Passed	2013-02-07T21:42:10+05:30
sriram-windows-remote-cbgd	upgrade-go-server	upgrade	Failed	2013-02-07T20:23:39+05:30
sriram-windows-remote-cbgd	transferFile2node	robocopy	Passed	2013-02-07T20:19:47+05:30
sriram-windows-remote-cbgd	transferFile	fetch-go-binaries	Passed	2013-02-07T20:18:17+05:30
sriram-windows-remote-cbgd	upgrade-go-server	upgrade	Passed	2013-02-07T16:21:02+05:30

For every job, the following columns are shown:

1. Pipeline: The pipeline to which the job belongs to
2. Stage: The stage to which the job belongs to
3. Job: The name of the job
4. Result: The result of the job - Passed, Failed, Cancelled or Rescheduled
5. Completed: The date when the Job completed
6. Duration: The duration that the Job took to finish - from scheduled till completed.

The job listing table can be sorted on any column, except for the Duration column.

## Using Agent details to debug agent issues

This page is useful to figure out if there are agent issues and hence a certain job keeps failing on that agent.

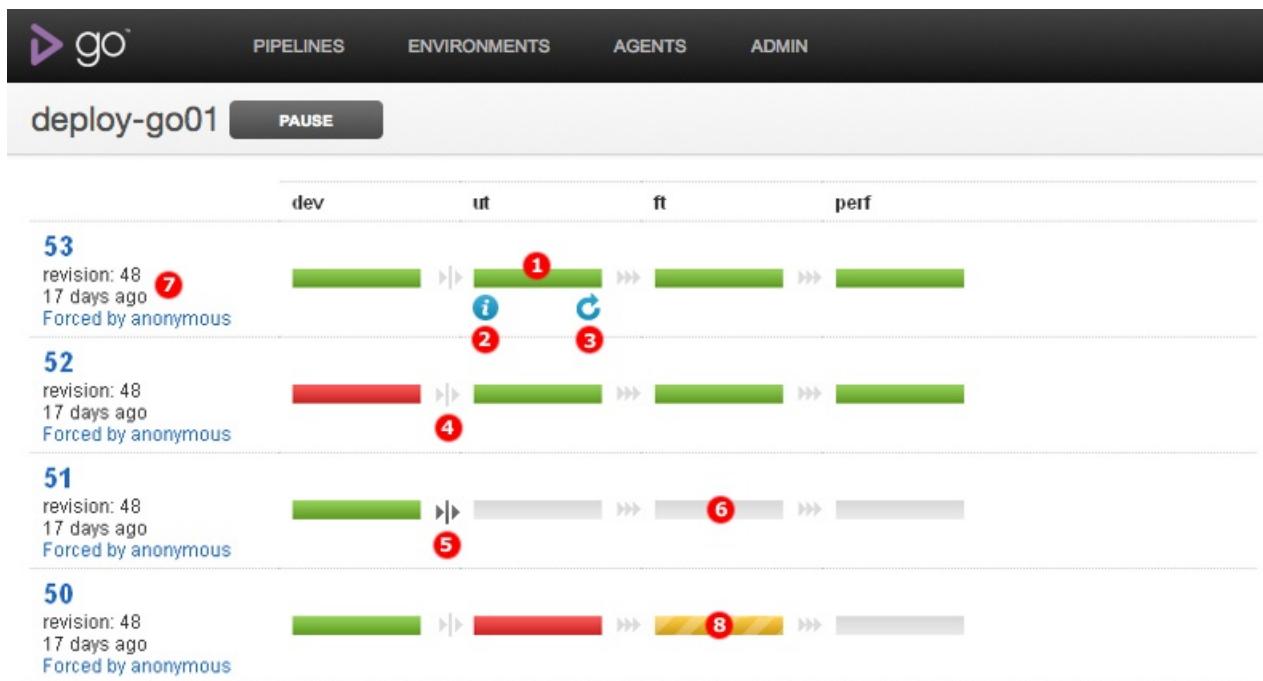
Consider a job which runs functional tests for a web application that need a browser to be available. The job was passing so far and only recently it has started to fail intermittently. Here are the steps you can follow to figure out if this is an agent issue.

1. Navigate to the [Job Details page](#) of the given job that failed.
2. Locate the "Agent" label and click on the link to the agent
3. Navigate to the "Job Run History" tab
4. Sort on the Job Name and locate the job that just navigated from

You'd notice that the job started to fail recently. You can even see if there are other jobs that have started failing around the same time by now sorting on the Completed date.

# Pipeline Activity

The Pipeline Activity page shows the history of stages for each pipeline label over the life of a specific pipeline.



## Key

1. Hover over a stage in the pipeline configuration box to see further options for that stage.
2. Click the info icon which appears on a stage on hover to show [stage details](#).
3. Use this button to re-run a particular stage. This option re-builds a stage and continues to build the subsequent stages thereon. This option is only available on a completed stage.
4. This indicates Approval that has already been triggered, either automatically or manually.
5. This indicates Approvals that have not yet been triggered. This could be the case where a stage needs manual approval or when a stage has failed.
6. This indicates a stage that has not yet been run.
7. The revision number and "modified by" provide a quick look at who activated this pipeline and why. Click "modified by" to show a list of comments and revisions.
8. An yellow color indicates that a stage is in progress.

**Also see...**

- Stage details
- Go overview

# Stage Details

The Stage Details page shows the details of a specific stage.

The screenshot shows the Stage Details page for a 'dev' stage in a 'cruise' pipeline. The top navigation bar shows 'cruise > 13.1.0.7794 > dev'. Below the navigation is a horizontal bar with 'dev' and 'dist' stages, where 'dev' has a red 'x' icon and 'dist' has a red upward arrow icon. A red arrow labeled '1' points to the 'dev' stage. To the right is a red RSS feed icon with a red arrow labeled '11' pointing to it.

Below the bar, a progress bar indicates 'Run: 1 of 1' with a red 'x' icon. Status information includes 'Cancelled' (with a red 'x' icon), 'Automatically triggered on 12 Feb, 2013 at 15:04:02 [+0530]', 'Duration: 01:01:23', and a 'Compare' link.

The main content area has tabs: Overview (selected), Pipeline Dependencies, Materials (highlighted with a red arrow labeled '5'), Jobs, Tests, Config, and Graphs. A red arrow labeled '6' points to the Pipeline Dependencies tab. A red arrow labeled '7' points to the Materials tab. A red arrow labeled '8' points to the Jobs tab. A red arrow labeled '9' points to the Tests tab. A red arrow labeled '10' points to the first failed job 'windows-10'. A red arrow labeled '11' points to the first material change 'Git - trunk'. A red arrow labeled '12' points to the stage name 'dev'.

The 'Materials' section shows 'Git - trunk' with revision 5f4ba67acd09ae6cbc57d4f7bba0bc6171ef7c03, modified by abc@abc.com on 2013-02-12T15:05:46+05:30, and comment #000. It also lists 'Mercurial - twist' with revision 83120bc8997cef75df1abb7a80724d5e1acf793a, modified by abc@abc.com on 2013-02-08T17:36:22+05:30, and comment:.

The 'Jobs' section shows a list of failed jobs: windows-19, windows-10 (highlighted with a red arrow labeled '10'), windows-11, windows-12 (highlighted with a red arrow labeled '4'), windows-13, windows-14, windows-15, windows-16, windows-17, and windows-18.

The 'Stage History' section shows five stage instances with their labels and IDs:

- 13.2.0.8037-53ce9ff02d5bdcc 2509736708250830e1d19372d
- 13.2.0.8036-e0d97fe07193644 d4a5f4fd466cb40b6aef76481
- 13.2.0.8035-28c33e83d74affe 5879d5429cee1df6e24c2bc4
- 13.2.0.8034-ab482cf16b43126 40441cec132992098265b8aab
- 13.2.0.8033-a2952808988638b 6b704ecfdd7a08bf33fca6ec4

## Key

1. Details of a specific stage run: run number, status, when it was triggered, who triggered it, duration of the stage
2. Name of the stage
3. The jobs in this stage are grouped based on status: Passed, Failed, Cancelled, In Progress. Expand these sections to see the jobs.
4. Failed jobs: Click the job name to view [job details](#) for that job.
5. Stage History shows the status and the pipeline label in which this stage has run. The latest 10 are shown by default. The rest are paginated, the user can select to view the details of this particular stage in any of the pipeline instances. This will indicate if the stage was a re-run and show the stage counter. Click on the stage instance in the stage history section to view the stage details page for that stage.
6. Displays a graphical visualization of the pipeline dependency chain.
7. Lists all the material changes that were part of the build in this stage.
8. Displays detailed information about the jobs in this stage.
9. Shows the failed build history for tests failing in the stage.
10. Cancelled job. Click the job name to view [job details](#) for that job
11. RSS feed for the stage in Atom format
12. Name of the stage

# Failed Build History

---

Results of test runs from jobs within a stage are aggregated up to the stage level. Failures are listed under the relevant pipeline instance label . Tests listed are ones that are failing in the stage instance currently being viewed. The tests are grouped by pipeline instance in which they started to fail (and are still failing). This gives you information about which users' checkins are responsible for test failures. On clicking the CHANGES link next to the Pipeline Label, the popup shows you the modifications to materials that have been built in this instance of the pipeline. All the stage instances till the time this stage was last seen green are listed in the failed build history. The pipelines are sorted by [natural ordering](#).

## Test Failures in the current stage

Other information that the Failed Build History section on the Stage Details page shows:  
(Image need annotation)

1. Total number of tests run
2. Total number of failures
3. Total number of errors
4. Failing test names grouped by the test suites in which they ran
5. Details link next to each of the job names which gives a popup with the failure/error message with a stack trace caused by the test
6. Users whose check-ins are responsible for the failing test in a given instance.
7. Pipeline labels where the currently failing tests started failing and are still failing in the instance being viewed.
8. The names of the jobs in which the test ran. Clicking on the job name will take you to the job details page.
9. Shows modifications which caused the stage instance to be triggered.
10. Shows the failure message and stack trace for the test failure/error for that job.

## Example 1

You are viewing stage 'Dev' of pipeline label '60'. The pipeline has been failing since label '59'. There are currently 4 failing tests. This is how they are listed.

- 60 has 3 failing test all of which started failing in 60
- 59 and 58 are listed because the 'dev' stage failed but none of the currently failing tests started failing because of the changes in 59 or 58. This could be because the tests that were failing in 59 got fixed by the checkins in 60. But these check-ins

broke other tests. Or this could be because none of the test ran in 59 and 58, an error occurred before the tests started running.

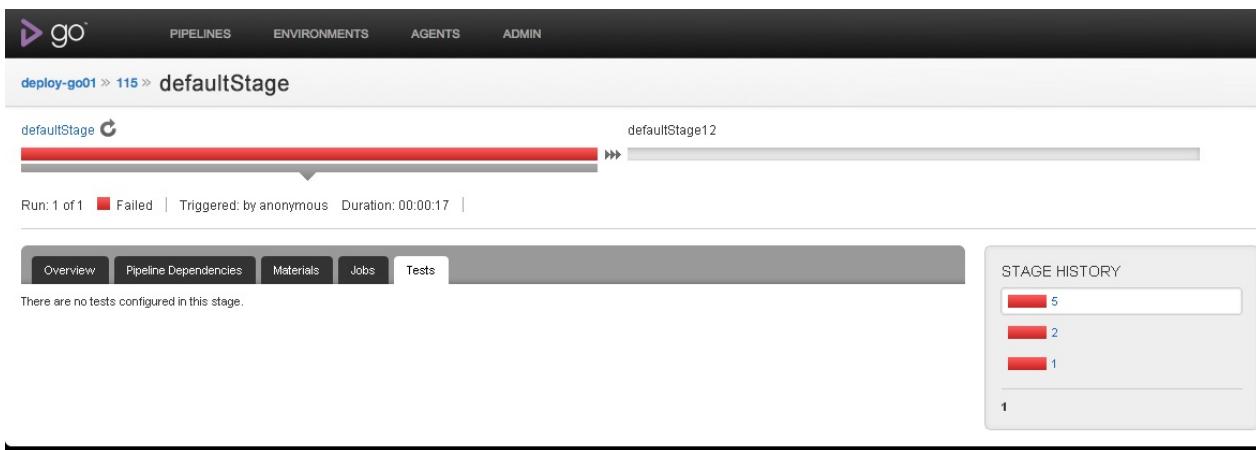
- The pipeline instances are listed in [natural order](#). In this case the schedule and natural order are the same.

## Example 2

You are viewing stage 'Dev' of pipeline label '59'. The pipeline has been failing since label '65'. There are currently 6 failing tests. This is how they are listed.

- The natural order of pipelines is 61, 60, 65, 59, 58, 57, 56, 55. This is the order in which they are listed.
- 65's changes caused 2 failing test which are still failing in 61 (instance being viewed).
- 60 had 1 new failing test which is still failing in 61.
- 61 has 3 newly failing tests.
- None of the currently failing tests started failing in 59, 58, 57, 56 or 55.

If there are no tests configured in the stage or Go is still computing results, this is the message that is displayed.



## Also See...

- [Job details](#)
- [Re-running job\(s\)](#)
- [Historical Configuration](#)

# Job Details

The Job Details page shows the details of a job within a specific stage.

The screenshot shows the Job Details page for a job named "app-UAT/53/deploy/1/P1.S1.Job1-05994646-320a-47a5-8ebd-30fede743a14". The top bar indicates the job is passed. Below it, the "Properties" tab is selected, showing a table of properties like cruise\_agent, cruise\_job\_duration, and cruise\_stage\_counter. Other tabs include Console, Tests, Failures, Artifacts, Materials, Coverage, and a custom tab labeled "Properties". A sidebar on the right lists recent jobs in a timeline, each with a status icon and a link to its details.

Property name	Property value
cruise_agent	blrstdcrspbs05.thoughtworks.com
cruise_job_duration	0
cruise_job_id	6673
cruise_job_result	Passed
cruise_pipeline_counter	28
cruise_pipeline_label	53
cruise_stage_counter	1
cruise_timestamp_01_scheduled	2010-06-29T21:36:30+05:30

## Key

1. The "Console" Tab shows the console output of the agent running the job in near-real time.
2. The "Tests" Tab shows junit compliant test output. (After you [upload test reports](#))
3. The "Failures" Tab shows any test, server, compilation, or network error associated with a failed job run.
4. The "Artifacts" Tab shows a collapsible list of the job's artifacts.
5. The "Materials" Tab shows what activated the pipeline through a list of comments and revisions.
6. The "Properties" Tab shows the list of [properties](#) for the job. Some properties are Go defaults. You can also automatically [save custom properties](#).
7. The "Coverage" Tab is an example of a custom tab. You can create [custom tabs](#) to view any uploaded artifact.
8. Click "Export property history to spreadsheet (csv)" to view a job's property values over time in an excel compatible format. This is useful for charting and graphing various build metrics.
9. The Jobs History sidebar lists recent jobs by date and time. Each job is highlighted with a passed, failed, or canceled icon. Click the job to view its Job Details information.

## Also See...

- Re-running job(s)

# Administration

There are four ways to configure pipelines etc

1. Via the admin UI described below
2. Direct XML edit via the admin UI's Config XML tab
3. Some limited configuration is possible via [config API](#)
4. Direct XML edit via the file system. By default, Go server polls the filesystem every 5 seconds for changes to cruise-config.xml. The location of this file is indicated in the top right corner of the Admin > Config XML tab.

## Pipelines

The "Pipelines" tab allows you to configure pipelines grouped in pipeline groups.

Pipelines    Templates    Source XML    Server Configuration    User Summary    OAuth Clients    OAuth Enabled Gadget Providers

Pipelines    + Add New Pipeline Group 1

Main\_Group    Edit    Delete 2

Pipeline	Actions
app_pipeline 5	Edit 6 Move to ▾ Clone 7 Delete 8 Extract Template
prod_pipeline	Edit Move to ▾ Clone Delete Extract Template

+ Create a new pipeline within this group 4

Empty\_group    Edit    Delete 3

No pipelines associated with this group

+ Create a new pipeline within this group

## Key

1. Add a new pipeline group
2. Edit the pipeline group name and permissions.
3. Delete an empty pipeline group.
4. Create a pipeline within a pipeline group
5. Click the pipeline name to select a pipeline to view or edit.
6. Click to view/edit a pipeline

7. Move pipeline to another pipeline group.
8. Delete a pipeline

## Pipeline Templates

The "Templates" tab allows you to configure pipeline templates which can be used to template pipelines.

Pipeline Templates [+ Add New Template](#) 1

**template-pipeline** [Edit](#) [Permissions](#) [Delete](#)

Pipeline	Actions
prod_pipeline	<a href="#">Edit</a> 4

**unused** [Edit](#) [Permissions](#) [Delete](#) 3

No pipelines associated with this template 5

## Key

1. Add a new pipeline template
2. Edit the pipeline template.
3. Delete an unused pipeline template.
4. Edit a pipeline using this pipeline template.
5. As a Go Administrator, you can now edit permissions for the template to make users [Template Administrators](#).

# Server Details

The Server Details page describes the Go server version and environment.

## Server Details

### Server Info

Go Server Version:	12.4.0(16415-c358c855a62a27)
JVM version:	23.2-b09
OS Information:	Linux 2.6.32-279.14.1.el6.x86_64
Usable space in artifacts repository:	1161666 Mb
Database schema version:	1301001
Go edition:	Enterprise

This box lists all the server information.

### Also see...

- [Go overview](#)

# Environments

The Environments page displays all environments, every pipeline and material associated with each environment, information on what is running in each environment, an icon informing you if there are new revisions, a button allowing you to deploy the latest revisions for each pipeline, a pipeline material breakdown, and a graphical status bar for each pipeline.

The screenshot shows the 'ENVIRONMENTS' tab selected in the top navigation bar. Below it, the 'Environments' section displays the 'AutoDeploy-Ubuntu9.10' environment. The interface includes:

- 1** A globe icon with a red notification badge containing the number '1'.
- 2** The environment name 'AutoDeploy-Ubuntu9.10'.
- 3** A graphical icon with a blue 'i' and a red notification badge containing the number '2'.
- 4** Pipeline label '2.0.0'.
- 5** A green progress bar indicating pipeline status.
- 6** A collapsed list icon with a red notification badge containing the number '6'.
- 7** A 'Compare' button with a red notification badge containing the number '8'.
- 8** Pipeline status message: '(triggered about 19 hours ago by cruise ) Passed: auto-upgrade'.
- 9** A table showing pipeline materials and their details:

Material	Revision	Check-in/trigger
cruise@10..../c	<b>8517475</b>	2 days ago
ruise_qa		
cruise	cruise/5079/dis t-all/1	about 19 hours ago

At the bottom are navigation buttons: a left arrow, a right arrow with a red notification badge containing the number '7', and a plus sign.

## Key

1. Name of the environment
2. Name of each pipeline associated with the environment
3. A graphical icon informing you if there are new revisions
4. The label name running in the environment for each pipeline, when the label was deployed and stage information
5. A graphical status bar of each pipeline broken down by stage and state
6. A collapsible list of all materials associated with each pipeline. The rows that are highlighted (colored and bold) indicate that there are new builds for those materials

that are yet to be deployed

7. Buttons to either deploy the latest revisions or specific revisions to an environment
8. Compare what changes have been deployed from a previous version

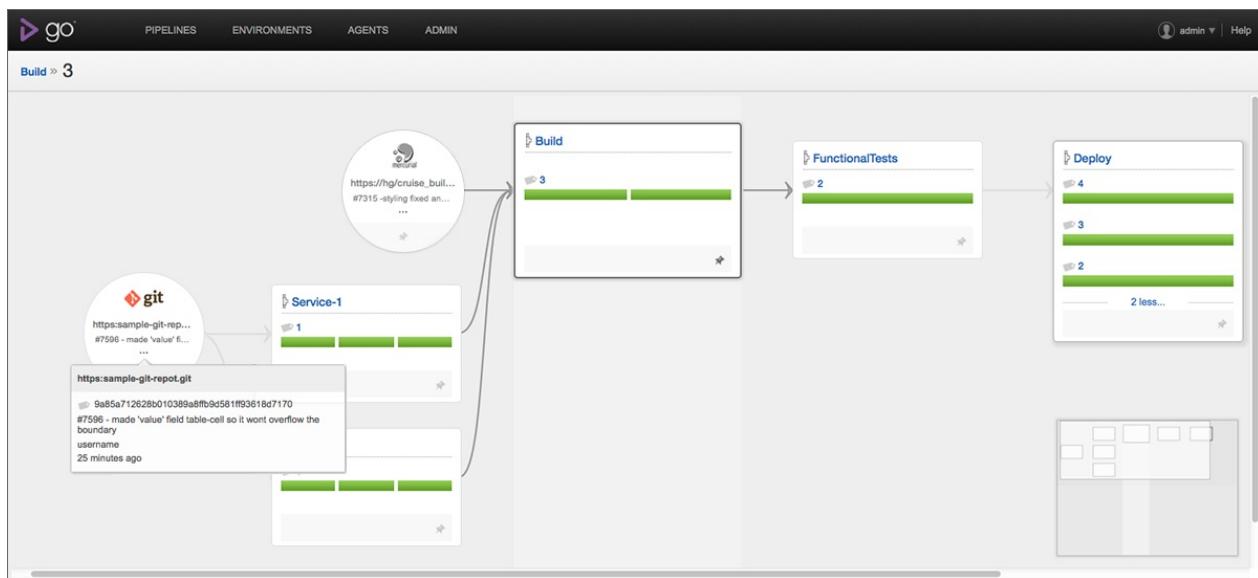
## Also see...

- [Managing Environments](#)

# Value Stream Map

## Introduction

Value Stream Map helps you visualize your CI/CD workflow. With a single click, it allows you to trace a commit from when it is checked in up to when it is deployed.



A value stream map can be drawn for every instance of a pipeline. It provides you with the ability to:

- See what caused the current pipeline to be triggered.
- See what downstream pipelines were triggered by the current pipeline.
- See the status of the current pipeline and all its upstream and downstream dependencies.
- See changes in dependencies of the pipeline across different runs of it.

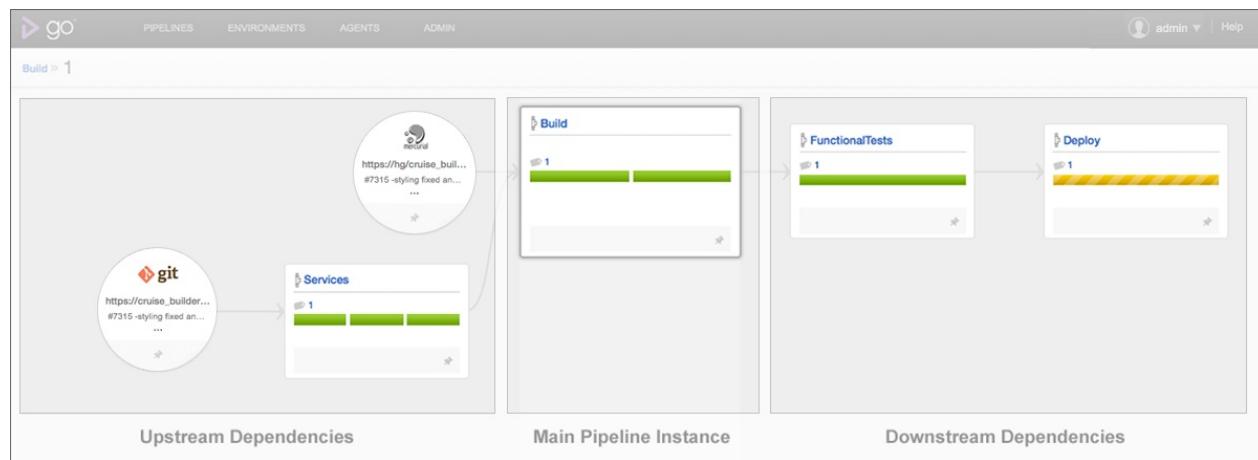
Along with all this, it also allows you to easily debug problems when a dependency/configuration change caused your build-test-release setup to break.

## Understanding the Value Stream Map

The Value Stream Map is laid out as an end-to-end dependency graph. The graph originates from source control materials and flows from left to right.

The pipeline instance for which the Value Stream Map is being viewed is the main

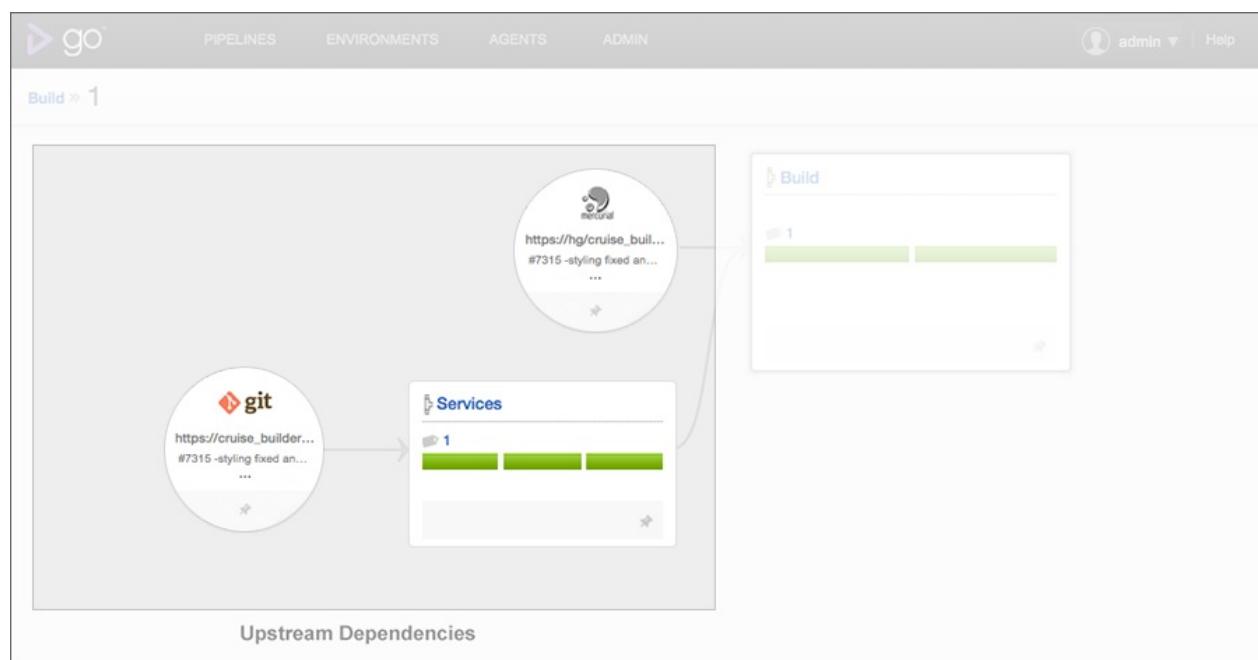
pipeline and is highlighted. Everything to the left of this pipeline are its upstream dependencies, ie all the materials that have contributed to this instance. Everything to the right are its downstream dependencies, ie, all the pipelines that it can potentially trigger or contribute to.



## Upstream Dependencies

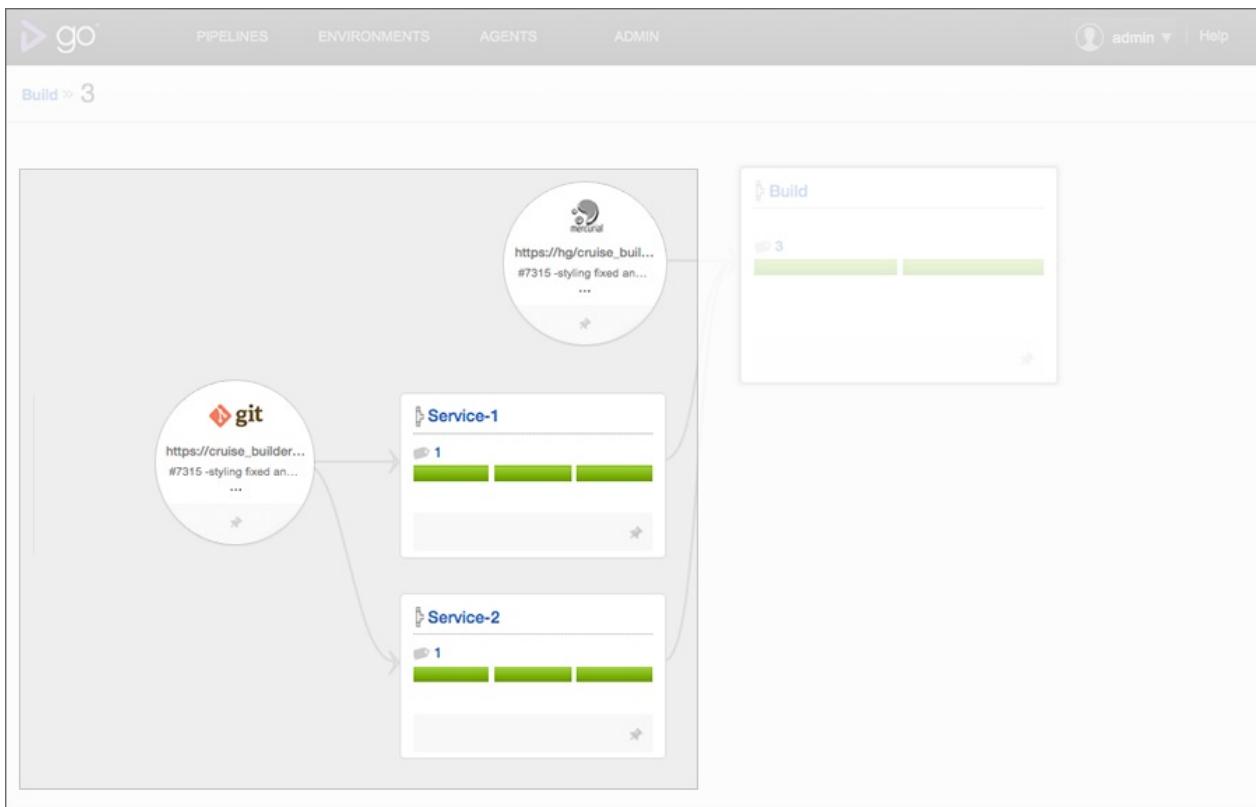
The upstream dependencies of the main pipeline are taken from history and show all the source control and pipeline dependency materials that have contributed to the main pipeline. Even when the Go Configuration changes after a certain instance of a pipeline, its upstream dependency graph will continue to reflect the older configuration with which it was run. This also means that it would display pipelines that do not exist in the configuration any more.

Let us assume that instance '1' of pipeline 'Build' is as below

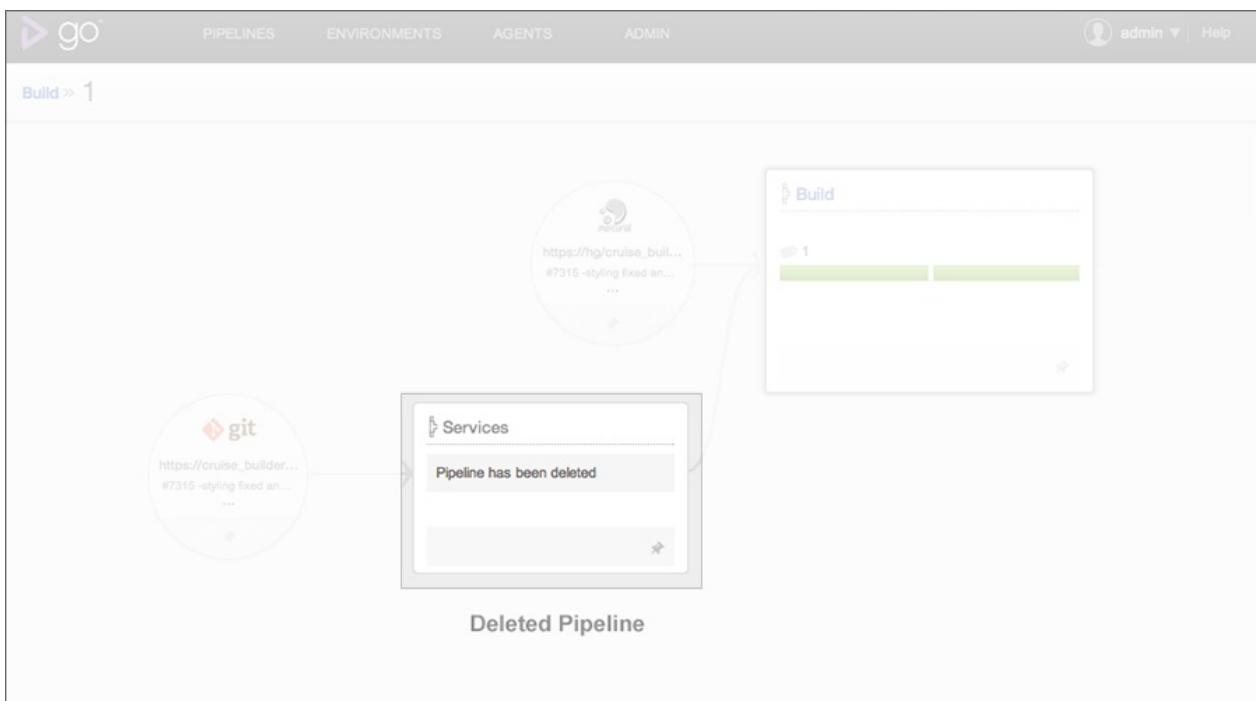


If the configuration changes to replace the pipeline 'Services' with 2 pipelines - 'Service-

'1' and 'Service-2', the next instance of pipeline 'Build' would reflect the change.

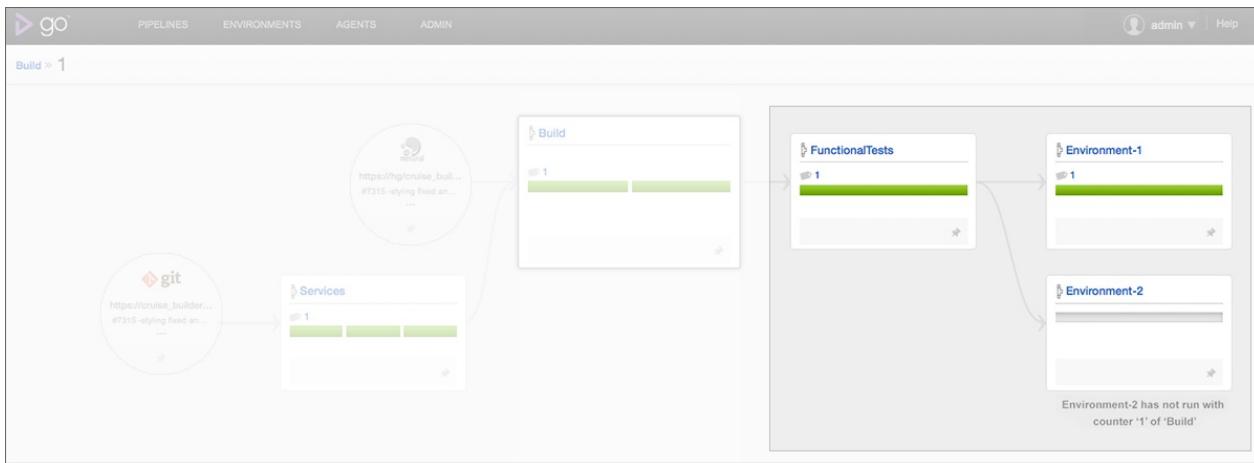


Value Stream Map of pipeline 'Build' with counter '1' would look as below

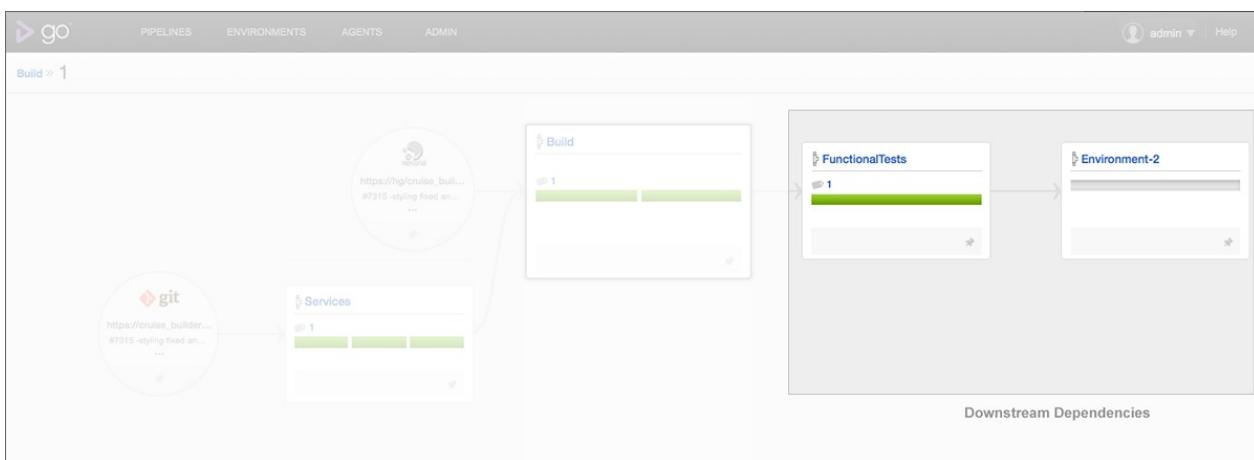


## Downstream Dependencies

The downstream dependencies of the main pipeline instance indicate what can happen with it. This information is always taken from the latest configuration. Pipelines that have not run yet are also shown

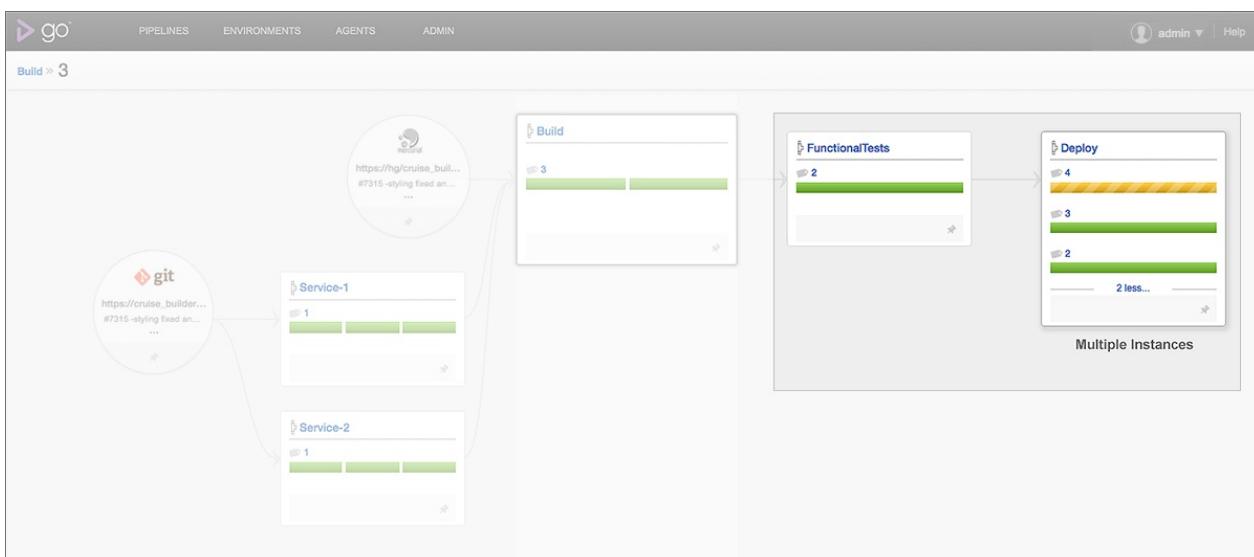


If Environment-1 is removed from the configuration, the Value Stream Map for the same instance of 'Build' would look as below.



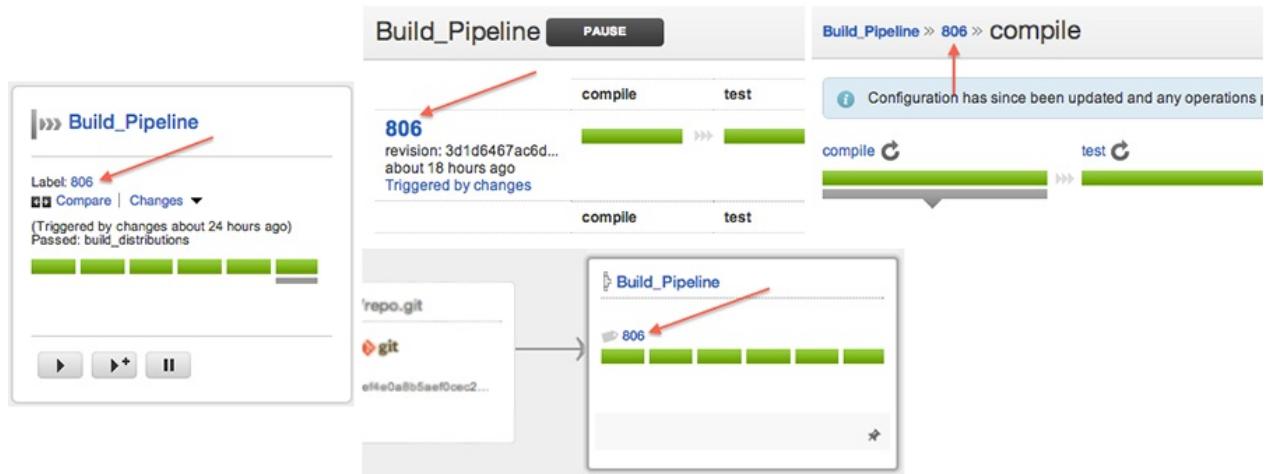
## Multiple Pipeline Instances

A pipeline could be re-triggered multiple times with the same revision. In such cases, all the instances are shown in descending order against that pipeline. In the below example, 'Deploy' has been triggered thrice with counter '2' of pipeline 'FunctionalTests'.

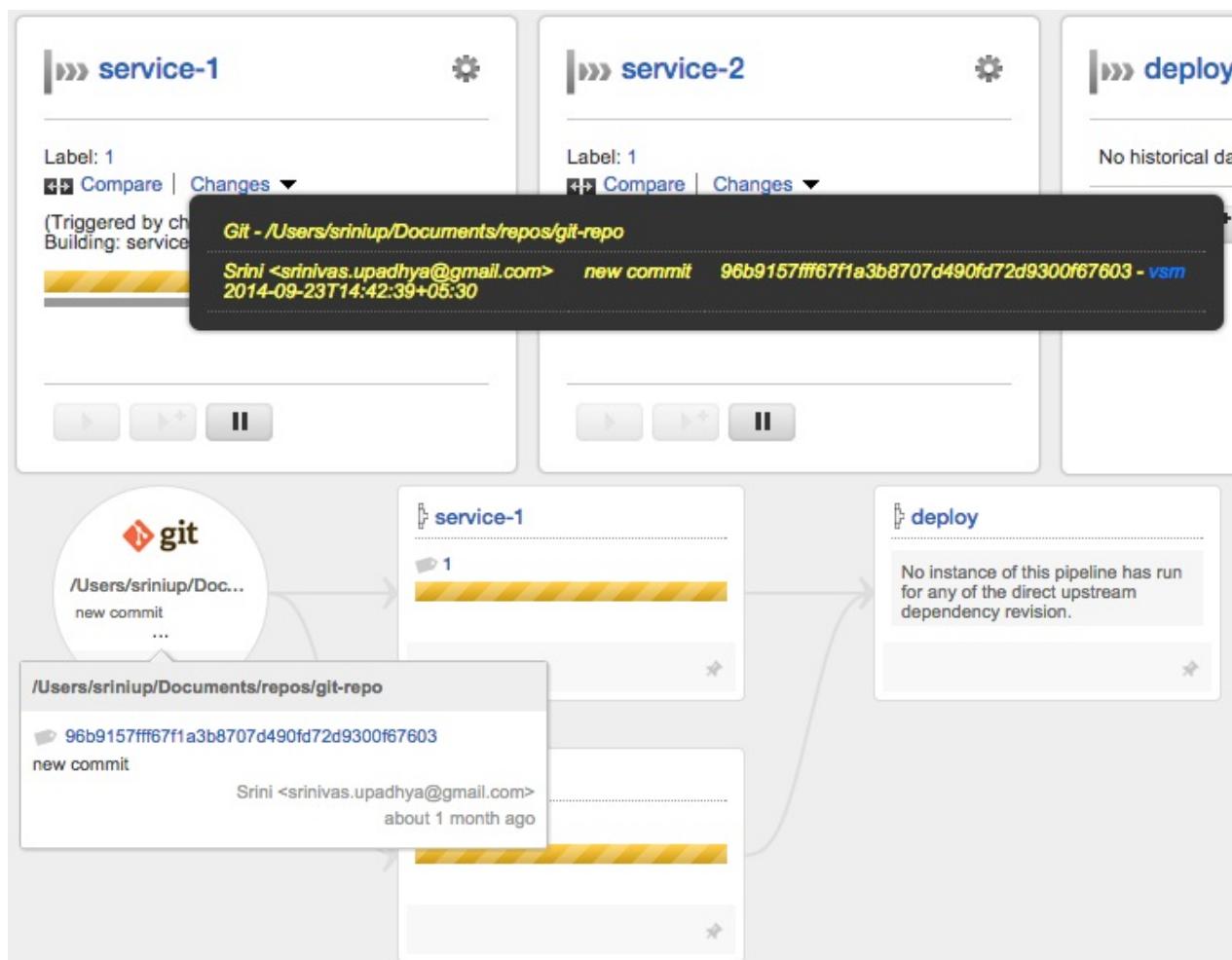


# Viewing the Value Stream Map

Every pipeline label in Go directs you to the value stream map of that instance of the pipeline.



You can access Value Stream Map for a commit in 2 ways:

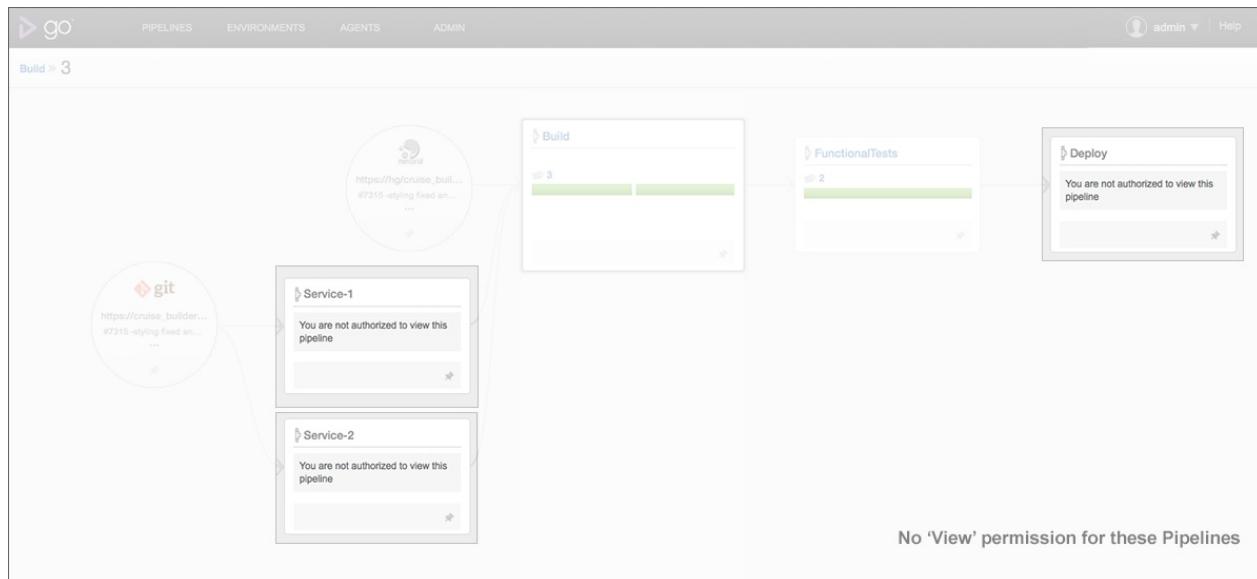


## Permissions

The permissions required to view a value stream map are as follows:

- Go Administrators have access to the value stream maps of all the pipelines that are configured.
- Users with view permissions for a pipeline will be able to view the value stream map for all instances of that pipeline.

However, there is one special case to be noted where the pipeline details might not be shown completely. If the user does not have view permissions for a pipeline in the Value Stream Map, its details, like the stages and instances run will not be shown.



If you are using Internet Explorer as your browser please note that Value Stream Map is supported with only versions 9 and above.

# **INSTALLING**

---

## **GO**

---

# System requirements

---

## Introduction

---

These requirements should meet the needs of most Go installations. You may need to allocate additional CPUs and/or memory on the machine hosting the Go Server if you intend to use a very large set of pipelines and/or agents.

## Client requirements

### Go works on the following browsers:

- Firefox
- Safari
- Chrome
- Internet Explorer 8, 9

## Go server requirements

### Windows

Go server is supported on Windows XP SP2+, Windows Server 2003, Windows Server 2008 and Windows 7.

- Java Runtime Environment (JRE) version 6 or above
- RAM: 1Gb minimum, 2Gb recommended
- 2 GHz (or higher)

### Mac OSX

- Mac OSX Leopard (10.5) and above
- Intel processor
- Apple Java Runtime Environment (JRE) version 6
- RAM: 1Gb minimum, 2Gb recommended

Go has limited support for Mountain Lion (OSX 10.8)

## **Linux**

We provide Debian packages which work on Ubuntu or Debian, and RPMs for RHEL, Fedora Core and CentOS. We support any OS based on Linux. ex: Ubuntu, Centos and RedHat Enterprise.

- Java Runtime Environment (JRE) version 6 and above
- RAM: 1Gb minimum, 2Gb recommended
- 2 GHz (or higher)

## **Solaris**

We provide Solaris packages which have been tested with Solaris 10 U5. They should work with OpenSolaris as well.

- Java Runtime Environment (JRE) version 6 and above
- RAM: 1Gb minimum, 2Gb recommended
- 2 GHz (or higher)

## **Extra requirements for Go server**

The host that runs your Go server should have a separate disk partition to store Go artifacts. The artifact repository can fill up quickly (especially if you are storing large binaries). If you don't create a separate partition for artifacts and your system disk fills up, Go and other applications on your system will behave unexpectedly. You are also likely to end up with corrupted data. Check the section on [Installing Go server](#) for more information on configuring your artifact repository.

Client software for your source code control tool must be installed on both your Go server and all Go build agents.

## **Go agent requirements**

## **Windows**

Go agent is supported on Windows XP SP2+, Windows Server 2003, Windows Server 2008 and Windows 7.

- Java Runtime Environment (JRE) version 6 and above
- RAM: 128Mb minimum, 256Mb recommended
- 2 GHz (or higher)

## **Mac OSX**

- Mac OSX Leopard (10.5) or higher
- Intel processor
- Apple Java Development Kit (JDK) version 6
- RAM: 128Mb minimum, 256Mb recommended

## **Linux**

We provide Debian packages which work on Ubuntu or Debian, and RPMs for RHEL, Fedora Core and CentOS. We support Ubuntu 7.10, Ubuntu 8.04, Centos 5.1 and RedHat Enterprise 5.

- Java Runtime Environment (JRE) version 6
- RAM: 128Mb minimum, 256Mb recommended
- 2 GHz (or higher)

## **Solaris**

We provide Solaris packages which have been tested with Solaris 10 U5. They should work with OpenSolaris as well.

- Java Runtime Environment (JRE) version 6
- RAM: 512Mb minimum, 1Gb recommended

## **Extra requirements for Go agent**

Go agent on its own does not require much memory or CPU. However, you need to ensure computers deployed as build agents have adequate resources to build your projects -- including sufficient disk space to check source code out of source control.

Client software for your source code control tool needs to be installed on all build agents. As well as, any other software required to build your application (if not accessed directly from the project source checked out from source control).

# Installing Go server

---

Before you install the Go server or agent, please take a look at [System Requirements](#).

## Installation

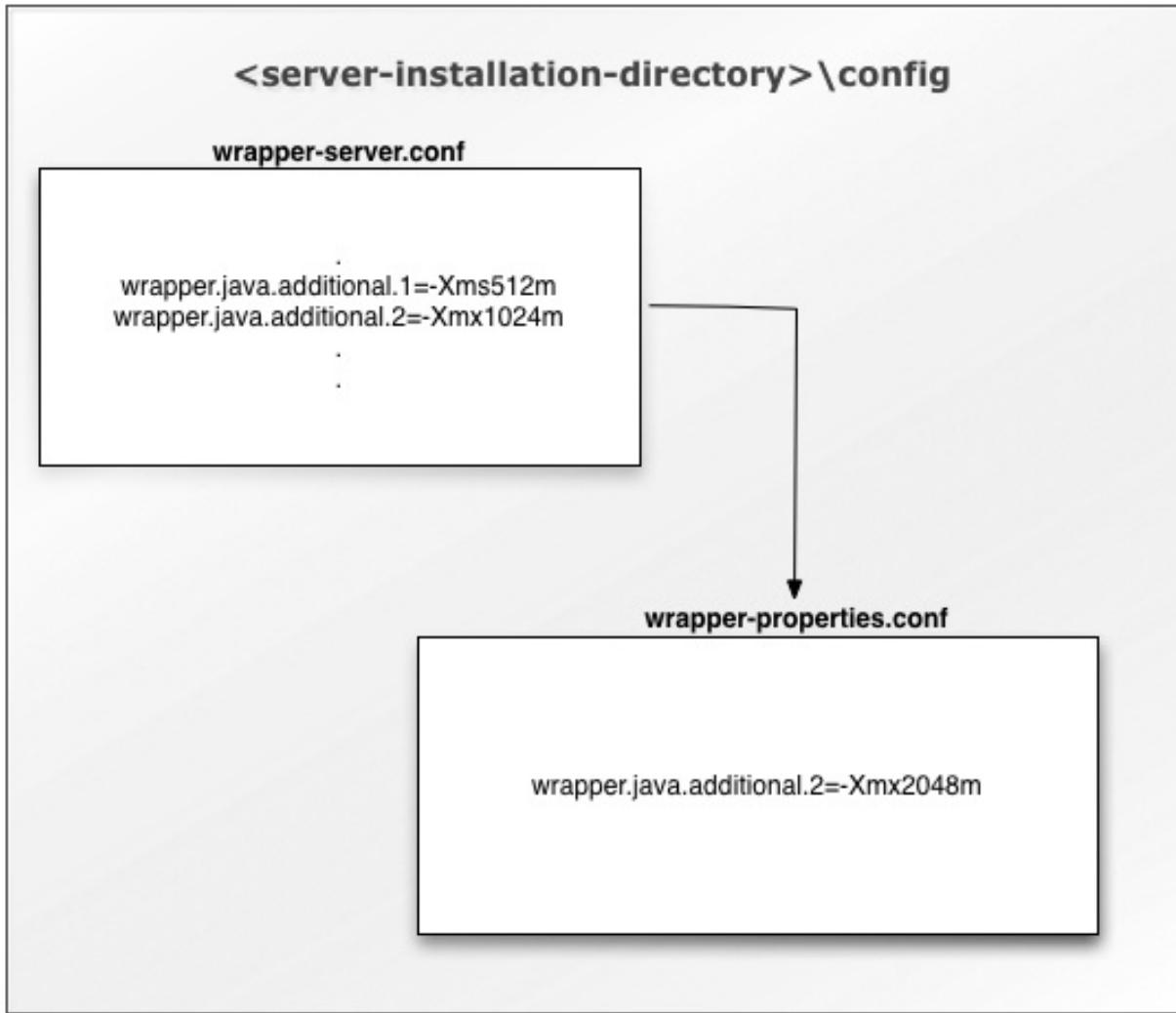
---

### How to install Go server for Windows

You must be logged in as a user with Administrator privileges to install the Go server on Windows.

1. Double-click the go-server-\${version}.exe installer file and follow the prompts to install Go.
2. During installation you will be asked to select a directory that will serve as the root path for your Go server installation. Go server will store all of its associated data in this directory by default.
3. You will next be prompted to choose the bundled Oracle 7 JRE or specify the location of JRE (or JDK) installed on your system
4. At the end of the installation, Go server will register itself as a windows service owned by 'Local System' and start running automatically.
5. Shortcuts to Go will be placed on your Desktop and in your Start Menu for convenience - double-click the shortcut to go to the Go dashboard.

### Override default startup arguments

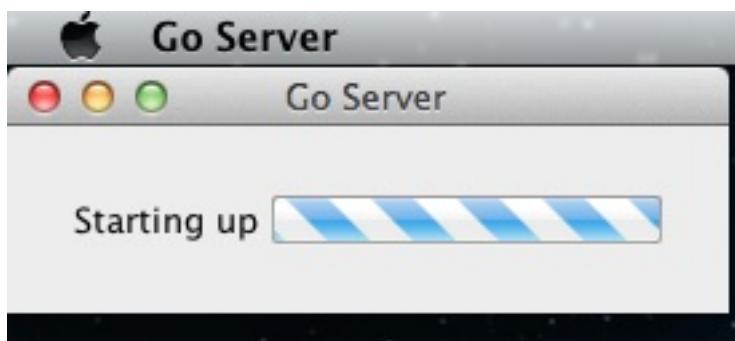


- Create a file named `wrapper-properties.conf` inside the `config` directory
- With reference to the representation above, if you wish to increase the maximum java heap size from default `1024m` to `2048m`,
  1. Copy the property `wrapper.java.additional.2=-Xmx1024m` from `wrapper-server.conf` to `wrapper-properties.conf`
  2. Change the value associated to `wrapper.java.additional.2` to the desired value `2048m` as shown in the above representation
- Adding a new property entails:
  1. Increment the **x** by 1 in `wrapper.java.additional.x` where **x** is the highest number in `wrapper-server.conf` and `wrapper-properties.conf` combined
  2. Add this newly created property to the `wrapper-properties.conf`

## How to install Go server for Mac OSX

1. Double-click the downloaded file to unzip the contents.
2. Drag the Go server Application to the Applications folder.
3. Go server will store its data in `Library/Application Support/Go Server` subfolder of

- the user's home folder
4. Double-click on the Go Server.app icon to open the launcher.
  5. While the Go server is starting up, you'll see a progress bar in the top left of your screen.



6. Once the Go server has started, it will open your default browser to the Go dashboard page.
7. To get back to the Go dashboard page when the server is running, click on the link in the About box of the Go server

Prior to 12.3.1, Go server stored its data in `/Library/Application Support/Go Server`. From 12.3.1, it will be in `<user-home>/Library/Application Support/Go Server`.

If you upgrade your Mac OS to Lion/Mountain Lion, Go installations prior to 12.3.1 will not continue to work. You will need to manually upgrade to 12.3.1 and copy the existing configuration from **/Library/Application Support** to **<user-home>/Library/Application Support**

On OSX 10.8.x (Mountain Lion), you may get the following error. **"Go Server" is damaged and can't be opened. You should move it to the Trash.** This is due to enhanced security protections. To allow the install to proceed

- Go to System Preferences->Personal->Security & Privacy.
- Launch the Security and Privacy applet.
- Click on the General tab to highlight it.
- Click on the lock icon to allow changes.
- Under the heading "Allow applications downloaded from:" click on the **Anywhere** radio button.

The installation will proceed as normal.

When it is finished, you can change the Security & Privacy setting back to the previous setting.

## How to install Go server for Linux

You must be logged in as root, or use *sudo*, to install Go on Linux. Go server also requires that Oracle or Open JRE or JDK - version 6 or above - is installed.

The Linux installer will create a user called *go* if one does not exist on the machine. The home directory will be set to */var/go*. If you want to create your own *go* user, make sure you do it before you install the Go server

### RPM based distributions (i.e. RedHat)

The Go server RPM installer has been tested on RedHat Enterprise Linux and CentOS. It should work on linux distributions which use rpms

- Run *rpm -i go-server-\${version}.noarch.rpm* to install Go server.

### Debian based distributions (i.e. Ubuntu)

The Go server deb installer has been tested on Ubuntu. It should work on linux distributions which use debs

1. Run *dpkg -i go-server-\${version}.deb* to install Go server.

The following command could be used after installation:

- Check Go server status with command *sudo /etc/init.d/go-server status*
- Start Go server with command *sudo /etc/init.d/go-server start*
- Stop Go server with command *sudo /etc/init.d/go-server stop*

Once the installation is complete the Go server will be started and it will print out the URL for the Dashboard page. This will be *http://< server host name >:8153/go*

## How to install Go server for Solaris

The Go server installer has been tested on OpenIndiana

You must be logged in as root, or use *sudo* or *pfexec*, to install Go under Solaris. Go server also requires that Oracle or Open JRE or JDK - version 6 or above - is installed.

The installer will create a user called *go* if one does not exist on the machine. The home directory will be set to */var/go*. If you want to create your own *go* user, make sure you do

it before you install the Go server.

1. Uncompress the package with the command `gzip -d go-server-${version}-solaris.gz`
2. Install the package with the command `pkgadd -d go-server-${version}-solaris`

The following command could be used after installation:

- Check Go server status with command `svcs go/server`
- Start Go server with command `svcadm enable -s go/server`
- Stop Go server with command `svcadm disable -s go/server`

## Copying existing config to a new Go-Server instance

You can replicate a go-server with all the pipeline, stage, job, tasks & materials definitions/configuration intact.

To do this Administrator should copy `cruise-config.xml` to the new server and clear 'serverId' attribute of server tag along with the license.

## Location of files after installation of Go server

---

### Windows

All the files for the Go server are under the root installation path on Windows. The default location is `C:\Program Files\Go Server`.

### Linux

```
/var/lib/go-server      #contains the binaries and database  
/etc/go                #contains the pipeline configuration files  
/var/log/go-server     #contains the server logs  
/usr/share/go-server   #contains the start script  
/etc/default/go-server #contains all the environment variables with default values.
```

### Mac OSX

```
< user-home >/Library/Application Support/Go Server
```

Some logging information is also written to `/var/log/system.log`

## Solaris

```
/var/lib/go-server    #contains the binaries and database  
/etc/go              #contains the configuration files  
/var/log/go-server   #contains the server logs  
/usr/share/go-server #contains the start script
```

## Also see...

- [Installing Go agents](#)
- [Configuring server details](#)
- [Configure Go to work with a proxy](#)

# Installing Go agent

---

## Introduction

---

You need to deploy at least one Go agent before you can build with Go. For the very simplest installation, you can run a Go agent on the same machine as your Go server.

## Installation

---

### Windows

You must be logged in as a user with Admin privileges to install the Go agent on Windows.

1. Double-click the go-agent-\${version}.exe installer file and follow the prompts to install Go.
2. During installation you will be asked to select a root path for your Go agent. In addition to holding your agent installation, this directory will contain the source code your agent checks out for every build.
3. You will next be prompted to choose the bundled Oracle JRE 7 or specify the location of JRE (or JDK) installed on your system
4. After installing the files, the installer will prompt you for the hostname or IP address of the Go server. If you leave this blank it will default to the local machine.
5. At the end of the installation, Go agent registers itself as a windows service and starts running automatically.

### Silent Installation

```
< agent installer > /S /SERVERIP=< ip of go server > /GO_AGENT_JAVA_HOME=< path to JI
```

SERVERIP is optional. Default value is localhost (127.0.0.1)

GO\_AGENT\_JAVA\_HOME is optional. Default is the packaged JRE 7

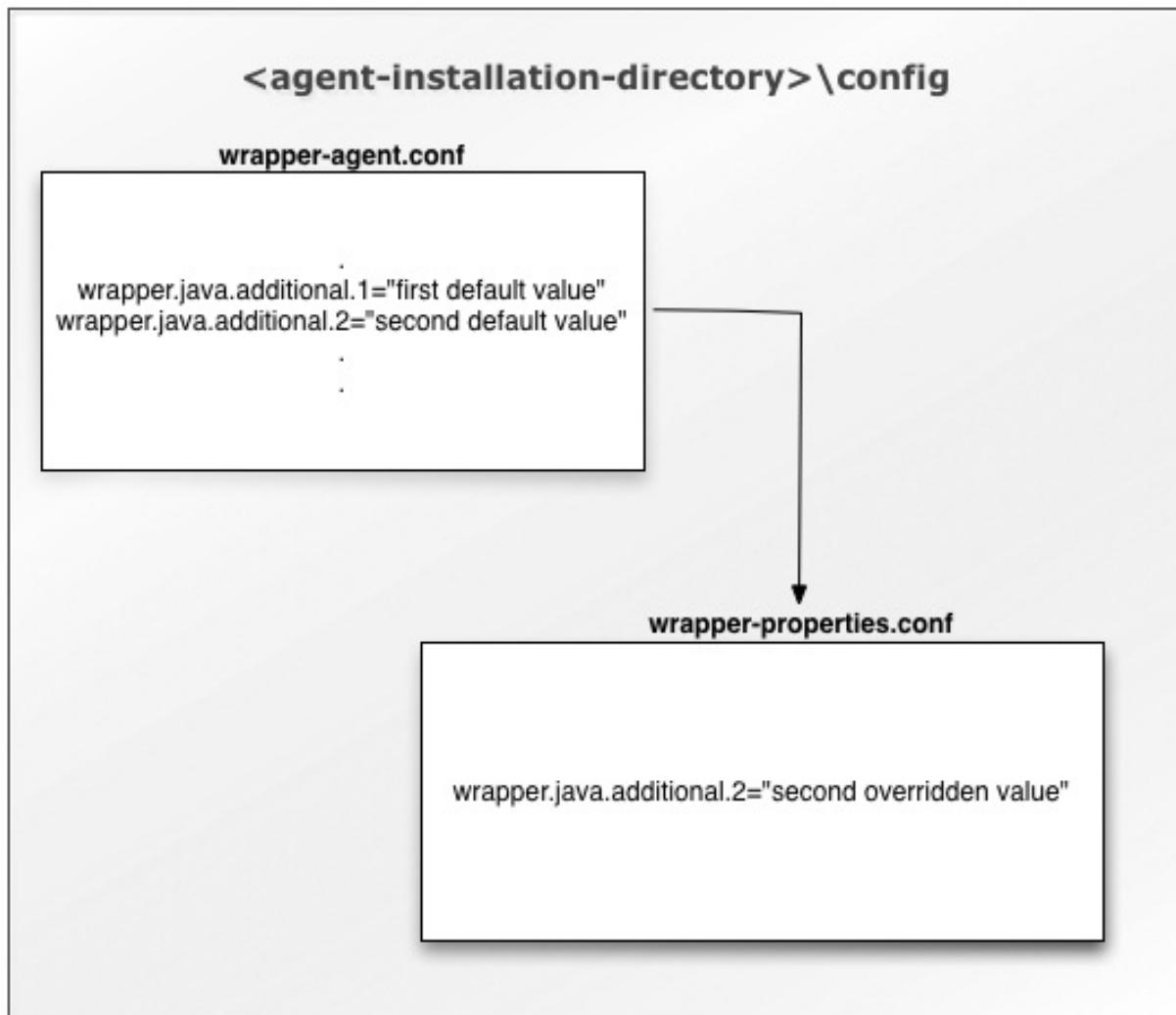
Installation-Directory is optional. Default value is C:\Program Files(x86)\Go Agent

- For example, C:\>go-agent-12.3.0-2000-setup.exe /S/SERVERIP=10.10.10.10 /D=C:\go\agent

If User Access Control feature is enabled on your Windows system, it needs to be turned off for silent installation to work

If you are using the silent installation to upgrade an agent, you should not specify the Installation-Directory option.

## Override default startup arguments

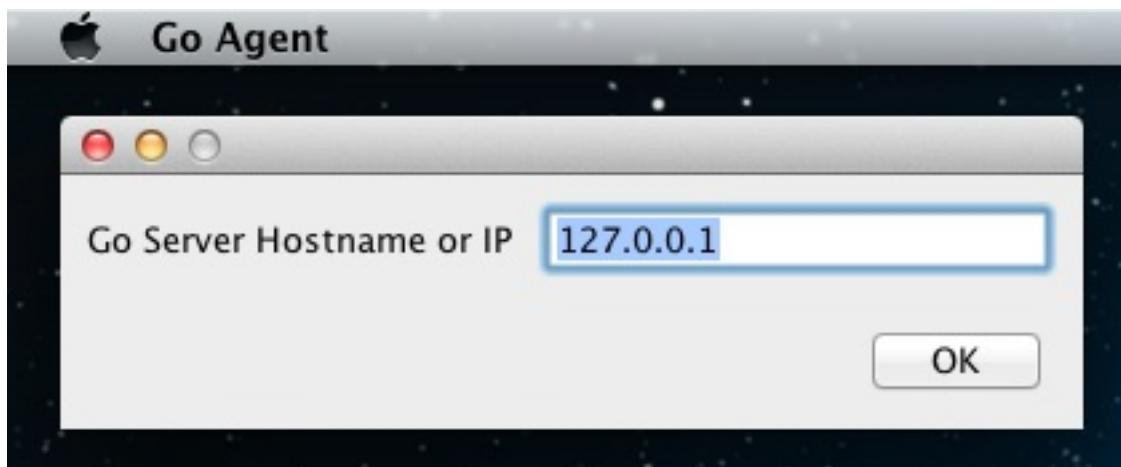


- Create a file named `wrapper-properties.conf` inside the `config` directory
- With reference to the representation above, if you wish to override `wrapper.java.additional.2`,
  1. Copy the property from `wrapper-agent.conf` to `wrapper-properties.conf`
  2. Change the value associated to `wrapper.java.additional.2` to the desired value
- Adding a new property entails:

1. Increment the **x** by 1 in *wrapper.java.additional.x* where **x** is the highest number in *wrapper-agent.conf* and *wrapper-properties.conf* combined
2. Add this newly created property to the *wrapper-properties.conf*

## Mac OSX

1. Double-click the downloaded file to unzip the contents.
2. Drag the Go Agent.app icon to the Applications folder. This will also be the directory where the agent checks out and builds the code.
3. Double-click on the Go Agent.app icon to open the launcher.
4. The very first time you run the Go agent on your machine you will be prompted for the hostname or IP address of your Go server. By default it will try connecting to the local machine. Click the OK button to continue.



On OSX 10.8.x (Mountain Lion), you may get the following error. **"Go Agent" is damaged and can't be opened. You should move it to the Trash.** This is due to enhanced security protections. To allow the install to proceed

- Go to System Preferences->Personal->Security & Privacy.
- Launch the Security and Privacy applet.
- Click on the General tab to highlight it.
- Click on the lock icon to allow changes.
- Under the heading "Allow applications downloaded from:" click on the **Anywhere** radio button.
- The installation will proceed as normal.
- When it is finished, you can change the Security & Privacy setting back to the previous setting.

## Linux

You must be logged in as root, or use *sudo*, to install Go on Linux. Go agent also requires that the Oracle or Open JRE or JDK - version 6 or above - is installed.

The installer will create a user called *go* if one does not exist on the machine. The home directory will be set to */var/go*. If you want to create your own *go* user, make sure you do it before you install the Go agent.

## RPM based distributions (ie RedHat)

The Go agent RPM installer has been tested on RedHat Enterprise Linux and CentOS. It should work on most RPM based Linux distributions.

- Run *rpm -i go-agent-\${version}.noarch.rpm* to install Go agent.

## Debian based distributions (ie Ubuntu)

The Go agent .deb installer has been tested on Ubuntu. However it should work on most Linux distributions which use debs.

- Run *dpkg -i go-agent-\${version}.deb* to install Go agent.

The following command could be used after installation:

- Check Go agents' status with command *sudo /etc/init.d/go-agent status*
- Start Go agents with command *sudo /etc/init.d/go-agent start*
- Stop Go agents with command *sudo /etc/init.d/go-agent stop*

Once the package has been installed you need to configure the hostname or IP address of your Go server and start the agent. To do this, do the following:

1. Open */etc/default/go-agent* in your favourite text editor.
2. Change the line *GO\_SERVER=127.0.0.1* to list the hostname or IP address of your Go server.
3. Save the file and exit your editor.
4. Run */etc/init.d/go-agent start* to start the agent.

## Solaris

You must be logged in as root, or use *sudo* or *pfexec*, to install Go on Solaris. Go agent also requires that Oracle or Open JRE or JDK - version 6 or above - is installed.

The installer will create a user called *go* if one does not exist on the machine. The home directory will be set to */var/go*. If you want to create your own *go* user, make sure you do it before you install the Go agent.

1. Uncompress the package with the command *gzip -d go-agent-\${version}-solaris.gz*
2. Install the package with the command *pkgadd -d go-agent-\${version}-solaris*
3. By default the agent will try connecting to localhost as the server. To change this, perform the following steps:
  - Open */etc/default/go-agent* in your favourite text editor.
  - Change the line *GO\_SERVER=127.0.0.1* to list the hostname or IP address of your Go server.
  - Save the file and exit your editor.
  - Run */etc/init.d/go-agent start* to start the agent.

The following command could be used after installation:

- Check Go agents' status with command *svcs go/agent*
- Start Go agents with command *svcadm enable -s go/agent*
- Stop Go agents with command *svcadm disable -s go/agent*

## Location of files after installing Go agent

---

### Windows

All the files for the Go agent are under its root installation folder in Windows, the default location is C:\Program Files\Go Agent.

### Linux

```
/var/lib/go-agent      #contains the binaries  
/usr/share/go-agent   #contains the start script  
/var/log/go-agent     #contains the agent logs  
/etc/default/go-agent #contains all the environment variables with default values. .
```

### Mac OSX

Some files for the Go agent are under its root installation folder in Mac OSX.

```
/Applications/Go Agent.app  
~/Library/Preferences/com.thoughtworks.studios.cruise.agent.properties
```

Some logging information is also written to `/var/log/system.log`

## Solaris

```
/var/lib/go-agent    #contains the binaries  
/usr/share/go-agent #contains the start script  
/var/log/go-agent   #contains the server logs
```

# Registering your agent with the server

---

For security reasons, all newly installed Go agents need to be enabled on the Go server before work is assigned to them. This prevents an unauthorized person from getting access to your source code. To enable a newly installed Go agent, do the following:

1. Open the Go server dashboard
2. Follow the instructions [here](#) to find the agent you've just installed on the list and add the agent to your cloud. The Go server will now schedule work for this agent.

# Running Go without installation

---

If you want to run Go on a platform which does not have a native installer or want to run Go without requiring to install it, you could do so by using the zip installers.

Ensure that the [system requirements](#) are met. Specifically:

- Java Runtime Environment (JRE) version 6 or above
- RAM: 1Gb minimum, 2Gb recommended
- 2 GHz (or higher)

## Run Go server

---

- Download the zip installer for Go server
- Unzip the installer in a folder of your choice. It creates a subfolder with the name *go-server-<version>*
- Set java in path
  - If you are on a Windows system, set *GO\_SERVER\_JAVA\_HOME* to the installation path of java on the system
  - If you are on a Unix system, set *JAVA\_HOME* to the installation path of java on the system
- Open a command prompt and go to the folder
- Start the server
  - If you are on a Windows system, run **start-server.bat**
  - If you are on a Unix system, run **server.sh**. (Ensure that *server.sh* is executable)

If you are on a system which does not support either command, alter the existing scripts suitably and use it

## Run Go agent

---

- Download the zip installer for Go agent
- Unzip the installer in a folder of your choice. It creates a subfolder with the name *go-agent-<version>*
- Set java in path
  - If you are on a Windows system, set *GO\_AGENT\_JAVA\_HOME* to the

installation path of java on the system

- If you are on a Unix system, set *JAVA\_HOME* to the installation path of java on the system
- Open a command prompt and go to the folder
- Start the agent
  - If you are on a Windows system, run **start-agent.bat**
  - If you are on a Unix system, run **agent.sh** . (Ensure that *agent.sh* is executable)

Go agent, by default, will attempt to connect to the Go server running on the same system. If you want it to connect to a different Go server, set the environment variable *GO\_SERVER* or edit the startup scripts suitably

If you are on a system which does not support either command, alter the existing script suitably and use it.

# Upgrading Go

---

## Introduction

---

To upgrade from a previous version of Go, it is only necessary to upgrade the Server. It is not necessary to stop or backup the Go Agents. Agents will automatically update to the correct version of Go.

## Before you start

Since Cruise 1.1 (legacy version of Go), we do not include a bundled version of the Subversion version control system. This means that if you use Subversion for your projects the server and all agents need to have Subversion installed and available on the system path.

Since Cruise 1.2 (legacy version of Go), we do not include a bundled version of ANT. This means that if you use ANT for your projects the server and all agents need to have ANT installed and available on the system path.

## Backing up your data

### Configuration Backup

As part of the configuration two files need to be backed up:

- Go's configuration is saved in the **cruise-config.xml** file
- Cipher file for password encryption.

Based on the OS your Go server is running on, both these files can be found at:

Operating System	Location
Linux	/etc/go
Windows	[Go install directory]\config
Mac OS X	< user-home >/Library/Application Support/Go Server

## Database backup

It is critical that the Go server be stopped before taking a backup of the database. If the Go server is not stopped, the backup may be corrupted. The database directory will be located at any one of the following locations based on what OS you're running on:

Operating System	Location
Linux	/var/lib/go-server/db
Windows	[Go install directory]\db
Mac OS X	< user-home >/Library/Application Support/Go Server/db

## Build Artifacts Backup

The Go server acts as a repository for all your build artifacts. While it is not essential to backup the artifacts before an upgrade, it is good practice to make regular backups of this directory.

You can configure where Go stores build artifacts. The following are the default locations of the artifacts if you have not customized its location:

Operating System	Location
Linux	/var/lib/go-server/artifacts
Windows	[Go install directory]\artifacts
Mac OS X	< user-home >/Library/Application Support/Go Server/artifacts

## Upgrading to the new version

You do not need to stop the Agents to perform an upgrade. Go agents will automatically update to the correct version of the software. You do not need to upgrade the Go agents. Any builds in progress will be rescheduled, and the existing pipelines will complete as expected.

If you are upgrading from a pre-2.1 release, the agent's directory structure will continue to be called "cruise-agent" and will not be renamed to "go-agent". This is normal and will not cause any issues.

On linux, you can ignore any permission errors logged as part of the upgrade of the cruise agent from a pre-2.1 release

Go will perform upgrades of its internal data structures when it starts. This process may take some time for installations with a large number of historical builds (10 to 15 minutes on very large installations). If you suspect that there is a problem with the upgrade, check the go-server.log to see if there are any reported errors. This is a one-time migration and subsequent restarts will be much faster.

## Windows

Run the Go installer. Make sure that you specify the same directory as your previously installed version.

If you have changed the Go Server Windows service to run as a different user, you will need to repeat this configuration change.

The installer will automatically start the service. Once Go completes its internal data changes, you should be able to see the Go webpage. Any existing Agents should automatically reconnect. Any builds in progress should continue, or be rescheduled.

## Linux

### **Debian based distributions (i.e. Ubuntu)**

Run the Go installer as described 'sudo dpkg -i [go-server-package-name]'.

### **RPM based distributions (i.e. RedHat)**

Run the Go installer as described 'sudo rpm -U [go-server-package-name]'.

## Macintosh OSX

The Macintosh OSX edition of Go does not support upgrades. You should follow the steps above to backup your data, uninstall Go/Cruise (by dragging the application into trash), and then perform a fresh installation.

## Solaris

The Solaris edition of Go does not support upgrades. You should follow the steps above to backup your data, uninstall Go/Cruise, and then perform a fresh installation.

## Notes

Use the notes from this section when upgrading to a particular version of Go.

## Version 2.3

As part of the 2.3 upgrade, the "dest" attribute of a material configured in a pipeline is case-insensitive. This would mean that if you have a pipeline P1 with two materials, say material M1 with dest = "foo" and material M2 with dest = "Foo", after the upgrade the dest folders will be automatically changed to M1 (dest = "foo\_(random\_string)") and M2 (dest = "Foo\_(random\_string)")

# Configuring server details

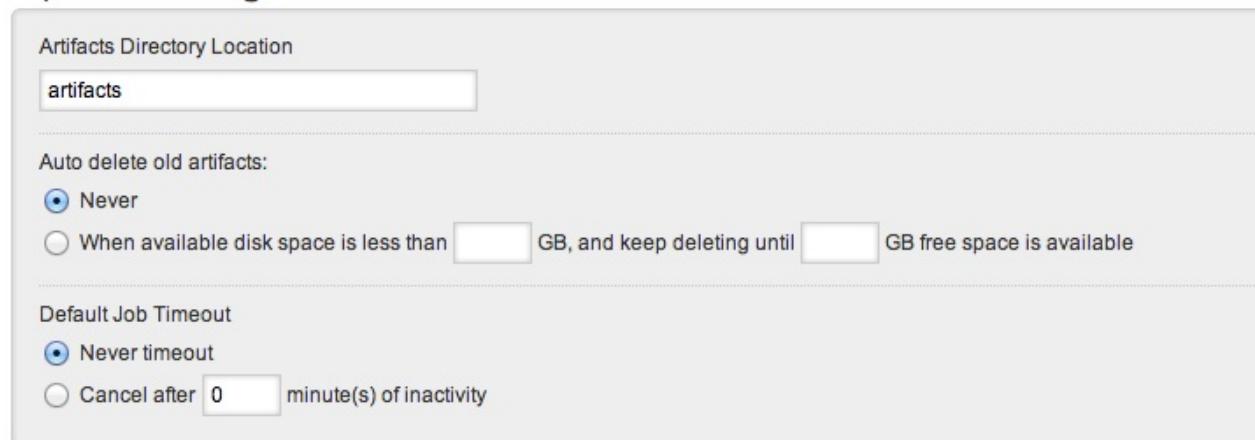
## Artifact repository configuration

Go needs no configuration once installed. However, we recommend that you create a separate partition on your computer's hard disk for Go server artifacts. The artifact repository can grow in size very quickly. If located on your system's main partition you may experience data loss and unpredictable application behaviour as the disk fills up.

Once you have created a new disk partition, you need to tell Go where to find it.

Click on "Server Configuration" tab of the "Admin" tab. Go to the "Pipeline Management" section.

### Pipeline Management



Specify the artifacts directory location and click on "Save"

Power users can also configure this via the **Config XML** tab on the Admin section:

```
<cruise>
  <server artifactsdir="/path/to/artifacts/directory">
    ...
  </server>
</cruise>
```

In Windows, you may need to assign your artifact repository partition a separate drive letter. In Windows, your configuration might look like this:

```
<cruise>
  <server artifactsdir="E:\go-artifacts">
    ...
  </server>
</cruise>
```

When you have entered this information, click "Save" to save the configuration file.

You can change the artifacts directory location at any time using the method described above, even when Go is running. However Go will not move existing artifacts to the new location for you, and changing the location while Go is running won't take effect until Go Server is restarted.

If you decide to move your artifact repository, the safe way to do it is:

1. pause all pipelines and wait until all active jobs on the agent grid has completed (all agents are in the state "idle")
2. shut down Go server
3. copy the artifact repository to the new location
4. edit Go's configuration file manually as described above to tell Go where to find the artifacts
5. restart Go server

Even when all active jobs on the agent grid have stopped, users may still be uploading artifacts using the RESTful URLs. This is why we need to stop Go server completely in order to be safe.

## Configure site URLs

---

Click on "Server Configuration" tab of the "Admin" tab. Go to the "Server Management" section.

# Server Management

The screenshot shows a user interface for managing server URLs. It features two input fields: 'Site URL' and 'Secure Site URL'. Each field has a placeholder text area and a question mark icon in the top right corner.

Go generates URLs that are relative to the base URL of the request. However, there are scenarios, such as sending emails, generating feeds where Go cannot rely upon publishing URLs relative to a request. If you have fronted Go with a reverse proxy, this value should be the base URL for the proxy and not the internal Go address. For this reason, it is necessary to specify this configuration. This URL should contain the port if your base URL contains a non-standard port.

Power users, if they so desire, can directly update the [server](#) section.

```
<cruise>
  <server siteUrl="http://<host>:<port>" secureSiteUrl="https://<host>:<secu...
    ...
  </server>
</cruise>
```

Certain features in Go, such as Mingle integration, require an HTTPS(SSL) endpoint. If you wish that your primary site URL be HTTP, but still want to have HTTPS endpoints for the features that require SSL, you can specify the `secureSiteUrl` attribute with a value of the base HTTPS URL.

## Also see...

- [Installing Go agents](#)
- [Configure Go to work with a proxy](#)
- [Displaying mingle gadgets in Go](#)

# Configure a Proxy

---

It is sometimes useful to front Go with a proxy server. In this section, we give you some tips and examples on how to achieve this.

## Go with Apache

---

An example of how to configure Go with Apache is shown below.

### Assumptions:

- You have Apache with mod\_proxy installed
- The Apache server sits on the same machine as the Go server (localhost)
- You want to enforce SSL connections

```
Listen nnn.nnn.nnn.nnn:80
NameVirtualHost nnn.nnn.nnn.nnn:80

<VirtualHost nnn.nnn.nnn.nnn:80>
    ServerName go.yourdomain.com
    DocumentRoot /var/www/html
    SSLProxyEngine on
    SSLEngine on
    ProxyPass / https://localhost:8154/
    ProxyPassReverse / https://localhost:8154/
</VirtualHost>
```

## OAuth 2.0 with Apache

---

If you have set up Go to use [OAuth 2.0 gadgets](#) and Go is fronted with an Apache server, then you have to set X\_FORWARDED\_PROTO to "https" in the https virtual host configuration section.

```
RequestHeader set X_FORWARDED_PROTO 'https'
```

This directive can replace HTTP request headers. The header is modified just before the content handler is run, allowing incoming headers to be changed to 'https'.

## Also see...

---

- [Configure site URLs](#)

# Performance Tuning

---

## Capacity Planning

---

This section provides recommendations to evaluate server hardware and memory requirements for your Go server. It also highlights some configurations which need to be taken care of when scaling Go.

### Minimum server requirements

The minimum requirements for a Go server can be found [here](#)

### Scaling Go

As the number of pipelines, agents and concurrent users increase in your setup, Go server may have to be scaled up by adding more memory and cores.

If you have questions or have custom requirements, please contact [support@thoughtworks.com](mailto:support@thoughtworks.com) to help with capacity planning for Go server

### Things to Remember

Do not run any other CPU intensive applications on the same box as the Go Server.

When the Go server is being scaled up to run with larger number of pipeline, agents and materials, ensure that the JVM has been allocated appropriate heap sizes. The default values for the Go server are -Xms512m (minimum) and -Xmx1024m (maximum). These values can be increased by setting the environment variables SERVER\_MEM (for minimum) and SERVER\_MAX\_MEM (for maximum).

On linux, these can be added/updated in /etc/default/go-server. On Windows, copy the following lines in [\*wrapper-properties.conf\*](#) and change it to desired value.

```
wrapper.java.additional.1=-Xms512m  
wrapper.java.additional.2=-Xmx1024m
```

For linux/unix users: If more than 100 agents are being used, an exception might be

seen in go-server.log mentioning "Too many open files". This may be an indication that there is a need to increase the number of file descriptors on the machine where Go Server is installed. On linux the command 'ulimit -n' can be used to check the total number of file descriptors. To bump up the total number for file descriptors user and system, follow these steps:

1. Edit /etc/security/limits.conf and add the lines: \* soft nofile 1024 \* hard nofile 65535
2. Edit /etc/pam.d/login, adding the line: session required /lib/security/pam\_limits.so
3. The system file descriptor limit can be increased by setting "fs.file-max" in the file "/etc/sysctl.conf". To set the limit to 65535 use echo "fs.file-max = 65535" >> /etc/sysctl.conf

## Tuning your JVM

Ensure that the latest JVM is used always, as there are major performance improvements with every release.

The minimum and maximum JVM heap space allocated to the Go server affects its performance. Go uses default values of 512m and 1024m for minimum and maximum JVM heap sizes respectively. However, for production environments, we recommend setting the minimum and maximum values to an identical value.

The default heap settings mentioned above are for a 32-bit JVM. But if the Go server is facing performance issues we recommend doubling the values in the heap settings and measuring performance. If it's seen that more than 3 GB of heap memory is needed, we recommend a switch to 64-bit JVM. Our tests show that Go server performs much better on a 64 bit JVM than a 32 bit JVM provided the heap memory has been increased appropriately. This is needed because 64-bit JVM makes use of 64-bit addresses instead of 32bits, allowing it to use more memory.

Start with the default settings and increase the heap memory incrementally to suit your application.

## Storage

For optimal performance in artifact transfer Go would need storage with good disk I/O throughput. We recommend local storage for Go database and artifacts. Disk space can be reclaimed through deletion of historical artifacts.

If using network storage is preferred, ensure that the speeds and throughput are good.

Use RAID Configuration for higher throughput if the Go Server is expected to be an intensive setup. If you expect to have large artifacts you could use different RAID configurations for Go database and artifacts. For example, 2 drives on RAID1 can be used for the Go database (for redundancy), 3+ hard drives on RAID5 can be used for artifacts so that access to database and artifacts is optimized.

## Improving Server Startup Time

The start up time for a very large Go Server instance could be improved by delaying material polling and pipeline scheduling to a few seconds after the server starts up. This would allow the server to warm up and cache some of the data before it is bombarded with threads that poll for material updates and pipelines that need to be scheduled.

Following are the JVM properties that enable such a delay:-

- `cruise.material.update.delay` - This value is specified in milliseconds. It has a default value of 10,000. This means that material polling would only start 10s after the server starts.
- `cruise.produce.build.cause.delay` - Likewise, this value is also specified in milliseconds. It again defaults to 10,000 meaning that scheduling of pipelines would take place only 10s after the server starts up.

The two values above do not affect the frequency of material polling or pipeline scheduling.

## Troubleshooting

---

### Enable GC Logging

An easy way to check the memory usage, heap size (initial and over time) and GC metrics of the application is by turning on GC logging. GC logging can be enabled using the following JVM arguments while starting the application (Note: the log file specified as file is reset each time the VM starts.)

```
-verbose:gc -Xloggc:file -XX:+PrintGCTimeStamps
```

In case of the Go server, these arguments will have to be added in the script that starts the Go jar:

- For linux : `/usr/share/go-server/server.sh`

- For Windows: {go\_server\_installion\_dir}/server.cmd. In most cases Go is installed in C:\Program Files\Go Server

## Using JConsole

JConsole is a graphical monitoring tool to monitor Java Virtual Machine (JVM) which comes as part of the JDK installation. It can be used to monitor the current state of a process without much overhead. If the Go server's performance is slow, some metrics can be immediately analysed using jconsole.

Since jconsole is a graphical tool, make sure you have an access to display, when running the following command. That is, use 'ssh -X' or VNC if Go is on linux. Use remote desktop if the Go server is on windows.

```
$ jconsole
```

Select the local process 'go.jar' when the jconsole GUI opens up. This shows the current heap memory usage, threads, cpu usage etc. Screenshots of the VM Summary and the overview page can be taken to be sent to the Go Support team, if required.

Please note that in case of linux, jconsole will have to be started as 'go' user. In Windows, starting the process as administrator should suffice.

More information about jconsole can be found [here](#).

## CPU and memory profiling

Yourkit java profiler is a recommended tool for profiling the CPU and memory of the GO Server.

To start using yourkit, download the latest version of the Yourkit java profiler from <http://www.yourkit.com/download/index.jsp>. Unpack to {yourkit\_profiler\_directory} The following steps will enable the Go server to pick up the yourkit profiler agent and enable us to take memory and cpu snapshots.

For Linux

1. Create a symlink for libyjpagent.so file to /usr/lib/yourkit folder. When the Go server starts up, it looks at this folder to see if it needs to start with profiling enabled or not. If you want to change the default path of the yourkit agent, you can edit server.sh at

```
/usr/share/go-server/server.sh
```

```
$ sudo ln -s {yourkit_profiler_directory}/bin/linux-x86-32/libyjagent.so /usr/l
```

For 64-bit JVM, the command is:

```
$ sudo ln -s {yourkit_profiler_directory}/bin/linux-x86-64/libyjagent.so /usr/l
```

2. Restart the server after this, and the yourkit agent should get picked up by the server VM. Let the server start up and agents get registered.

For Windows

1. By default, Go server looks for the yourkit profiler agent yjagent.dll in the location C:\yjagent.dll. Therefore, copy the file yjagent.dll (which is the yourkit profiler agent) from '{yourkit\_profiler\_directory}\bin\win32' to C:\yjagent.dll. Copy the file from '{yourkit\_profiler\_directory}\bin\win64' if you are using 64 bit JVM.
2. To change the above mentioned default location: define environment variable YOURKIT\_PATH with value equal to location of yjagent.dll.
3. If you are running the Go server as a service, you will need to perform an additional step. In the config folder of the Go server installation, edit the *wrapper-properties.conf* file, and add an additional property with the following value

```
"-agentpath: < Path to yjagent.dll >=port=6133,builtinprobes=none"
```

For example, if there are 16 properties already defined, add this 17th property as shown below

```
wrapper.java.additional.17="-agentpath:C:\yjagent.dll=port=6133,builtinprobes=n
```

Use the following steps to take profile the application and take snapshots. The {hostname} mentioned here is the hostname of the Go Server. In most cases, it would be 'localhost'. The value of {port} is 6133, because Go starts the yjagent on port 6133.

## 1. To start profiling, run:

```
$ java -jar {yourkit_profiler_directory}/lib/yjp-controller-api-redist.jar {host}
```

```
$ java -jar {yourkit_profiler_directory}/lib/yjp-controller-api-redist.jar {host}
```

If memory allocation profiling is also required:

```
$ java -jar {yourkit_profiler_directory}/lib/yjp-controller-api-redist.jar {host}
```

2. Let the server run for some time till you start seeing performance problems. 30 mins of snapshot should give us enough data.

3. To capture the snapshot - Run:

```
$ java -jar {yourkit_profiler_directory}/lib/yjp-controller-api-redist.jar {host}
```

To capture memory snapshot

```
$ java -jar {yourkit_profiler_directory}/lib/yjp-controller-api-redist.jar {host}
```

4. To stop profiling, run:

```
$ java -jar {yourkit_profiler_directory}/lib/yjp-controller-api-redist.jar {host}
```

```
$ java -jar {yourkit_profiler_directory}/lib/yjp-controller-api-redist.jar {host}
```

If memory profiling was turned on, stop it using the following command

```
$ java -jar {yourkit_profiler_directory}/lib/yjp-controller-api-redist.jar {host}
```

5. Once you're done profiling, run the following so that on the next Go server restart, the agent is not loaded into the JVM.

In case of linux, run the following command:

```
$ sudo rm /usr/lib/yourkit/libyjpagent.so
```

In case of windows, delete the file C:\yjpagent.dll. If you were using the variable YOURKIT\_PATH, then remove the environment variable.

These snapshots will be saved in the yourkit configured snapshots folder. They can be sent to the Go Support so that they can be examined to help find the root cause of the performance.

## Contact Go Support

If the Go server continues to behave poorly, send us the following data.

1. Database file cruise.h2.db. Stop the server and take a backup of the database.  
Location:

Linux: /var/lib/db/h2db/cruise.h2.db

Windows: {go\_installation\_dir}\db\h2db\cruise.h2.db

2. Log file go-server.log. Location:

Linux: /var/log/go-server/go-server.log

Windows: {go\_installation\_dir}\go-server.log

3. Go config file cruise-config.xml. Location:

Linux: /etc/go/cruise-config.xml

Windows: {go\_installation\_dir}\config\cruise-config.xml

4. If any Yourkit and jconsole snapshots as mentioned in the previous points, its useful if that can be sent too.

# **CONFIGURING**

---

## **GO**

---

# Setup a new pipeline

---

## New Pipeline Wizard

---

After you've entered your license information, clicking on the **Pipelines** tab will take you to the "Add new pipeline" page. You can also add a pipeline by navigating to the Admin page and clicking on the "Create a new pipeline within a group" link. You can create a pipeline in 3 steps.

### Step 1: Basic Settings

Add Pipeline

Step 1: Basic Settings    Step 2: Materials    Step 3: Stage/Job

Basic Settings

Pipeline Name\*  1

Pipeline Group Name  2

\* indicates a required field

CANCEL    NEXT



1. Fill in the pipeline name
2. Fill in the pipeline group

### Step 2: Material

Add Pipeline

Step 1:  
Basic Settings      Step 2:  
Materials      Step 3:  
Stage/Job

Materials

Material Type\*  
 1

URL\*  
 2

Branch

Poll for new changes

**CHECK CONNECTION**

CANCEL   PREVIOUS   NEXT

1. Choose the material type. The material can be your Source Control Management (SCM or version control) repository or another pipeline or a [package repository](#) (e.g. [yum](#)). Currently Go supports the following SCMs:
  - i. Subversion
  - ii. Mercurial
  - iii. Git
  - iv. Team Foundation Server.and the package repository.
2. Fill in settings specific to the material type.

## Step 3: Stage and Job

**Step 1:**  
Basic Settings

**Step 2:**  
Materials

**Step 3:**  
Stage/Job

## Stage/Job

Configuration Type  Define Stages  Use Template

Stage Name\*

defaultStage

1

Trigger Type:  On Success  Manual [?](#)

\* indicates a required field

### Initial Job and Task

You can add more jobs and tasks to this stage once the stage has been created.

Job Name\*

defaultJob

2

Task Type\*

Ant

3

Build file

[?](#)

Target

[?](#)

Working directory

[?](#)

A pipeline contains one or more stages. Define the first stage of your pipeline

1. Fill in the Stage name.
2. Fill in the Job name.

3. Fill in the task type and the command for the task.
4. If you use Ant, NAnt or Rake for scripting, Go provides convenience wrappers for these tools. To use any other scripting tool (e.g: Maven, msbuild, etc.), choose the "More..." option to use the [command repository](#) or specify the command line syntax for that tool.

See the [Managing pipelines](#) documentation for editing these settings following the creation of your pipeline.

## Initial task settings

---

### Ant

The Ant task allows you to run an ant script. Go does not include Ant and so you must ensure that it is already on the command path. By default it will use build.xml in the agent's working directory as the build file. If you want to customize the build file or build target, click the [edit](#) link to change the defaults.

For this option to work, Ant needs to be installed on the Go Agent(s) and the *go user* should be able to execute it.

### NAnt

The NAnt task allows you to run a NAnt script. Go does not include NAnt and so you must ensure that it is already on the command path. By default it will use default.build as build file in the agent's working directory. If you want to customize the build file or build target, click the [edit](#) link to change the defaults.

For this option to work, NAnt needs to be installed on the Go Agent(s) and the *go user* should be able to execute it.

### Rake

The Rake task allows you to run a ruby rake build. Go does not include ruby or rake and so you must ensure that it is correctly installed on the agents. Go will assume the standard **rakefile** exists in the working directory of the agent.

For this option to work, Rake needs to be installed on the Go Agent(s) and the *go user* should be able to execute it.

## More...

In addition to the above tasks, Go allows you to run anything on the command line. You can use the [command repository](#) to help you choose the command. Alternately you can specify a command on your own.

You can see the complete configuration reference [here](#).

## Also See

- [Adding a material to an existing pipeline](#)
- [Adding a stage to an existing pipeline](#)
- [Adding a job to an existing pipeline](#)
- [Role-based authorization](#)

# Managing pipelines

Go can be configured using [Administration Tab](#). You can perform operations like add/edit Pipelines, Stages, Jobs, Tasks, Templates and Pipeline group. You can also configure Go by editing the full XML file if you wish, by clicking on the **Config XML** section of the [Administration](#) tab. Go will check the syntax of the configuration before it saves it again

## Creating a new pipeline

To create a new pipeline, go to the **Pipelines** sub-tab of the [Administration](#) tab and click on the "[Create a new pipeline within this group](#)" link as shown in the screen shot below.



## Add a new material to an existing pipeline

Now that you have a pipeline, lets add another material to it.

- Navigate to the new pipeline you created by clicking on the **Edit** link under the Actions against it. You can also click on the name of the pipeline.



- Click on the Materials tab.

## » new\_pipeline

General Options   Project Management   Materials   Stages   Environment Variables   Parameters

### Basic Settings

Pipeline Name\*  
new\_pipeline

Label Template\*  
\${COUNT} 

Automatic pipeline locking   
 Automatic pipeline scheduling 

- You will notice an existing material . Click on the "Add new material" link.

» new\_pipeline

General Options   Project Management   Materials   Stages   Environment Variables   Parameters

### Materials

Name	Type	URL	Remove
https://svn-repo/svn/test/	Subversion	https://svn-repo/svn/test/	

- You will get the following message

Add Material - Subversion 

 In order to configure multiple materials for this pipeline, each of its material needs have to a 'Destination Directory' specified.  
Please edit the existing material and specify a 'Destination Directory' in order to proceed with this operation.



- Edit the existing material and specify the destination directory

## Edit Material - Subversion



Material Name  
 (?)

URL\*

Poll for new changes

Username

Password  
  Change Password

Check Externals

Destination Directory

SAVE CANCEL

- Click "Save".

## Blacklist

---

Often you do want to specify a set of files that Go should ignore when it checks for changes. Repository changesets which contain only these files will not automatically trigger a pipeline. These are detailed in the [ignore](#) section of the [configuration reference](#).

- Enter the items to blacklist using ant-style syntax below

**Edit Material - Subversion**

Password  
  Change Password

Check Externals

Destination Directory

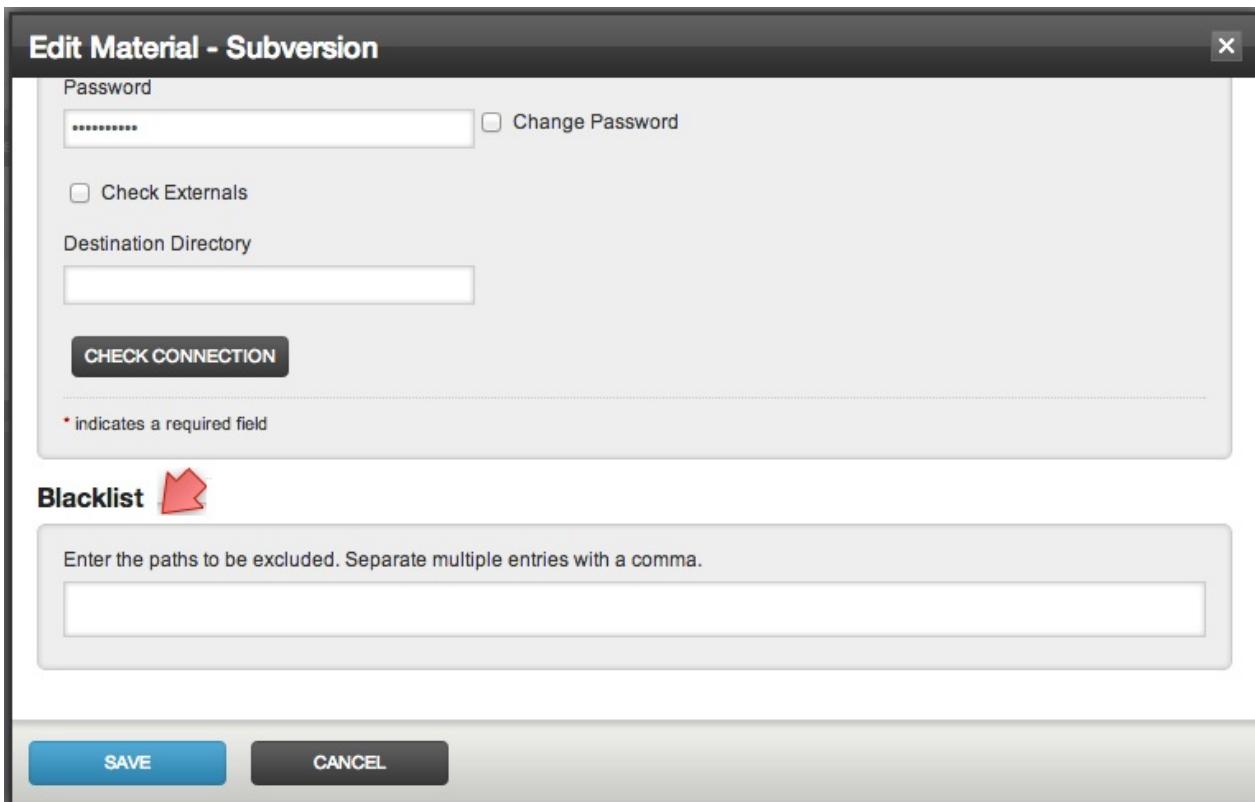
**CHECK CONNECTION**

\* indicates a required field

**Blacklist** 

Enter the paths to be excluded. Separate multiple entries with a comma.

**SAVE** **CANCEL**



- Click "Save".

## Add a new stage to an existing pipeline

Now that you have a pipeline with a single stage, lets add more stages to it.

- Navigate to the new pipeline you created by clicking on the **Edit** link under the Actions against it. You can also click on the name of the pipeline.

Pipelines  Add New Pipeline Group

 my\_group  Edit 

Pipeline	Actions
new_pipeline 	 Edit  Move to  Clone  Delete 



- Click on the Stages tab.

## » new\_pipeline

General Options   Project Management   Materials   Stages   Environment Variables   Parameters

### Basic Settings

Pipeline Name\*  
new\_pipeline

Label Template\*  
\${COUNT} 

Automatic pipeline locking   
 Automatic pipeline scheduling 

- You will notice that a defaultStage exists. Click on the "Add new stage" link.

## » new\_pipeline

General Options   Project Management   Materials   Stages   Environment Variables   Parameters

### Stages

Configuration Type  Define Stages  Use Template

Order	Stage	Trigger Type	Jobs	Remove
	defaultStage 	On Success	1	
<a href="#">+ Add new stage</a>				

- Fill stage name and trigger type.
- Fill in the details for the first job and first task belonging to this job. You can [add more jobs](#) and [add more tasks](#) to the jobs.
- Click on help icon next to the fields to get additional details about the fields you are editing.

## Stage Information

Stage Name\* 

Trigger Type:   On Success  Manual

\* indicates a required field

## Initial Job and Task

You can add more jobs and tasks to this stage once the stage has been created.

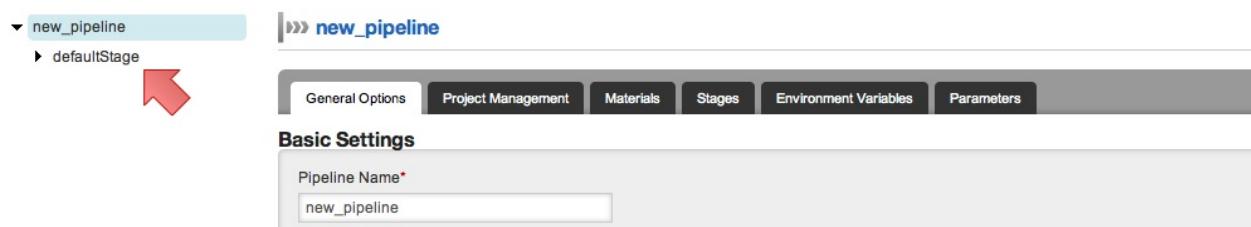
Job Name\* 

- Click "Save".

# Add a new job to an existing stage

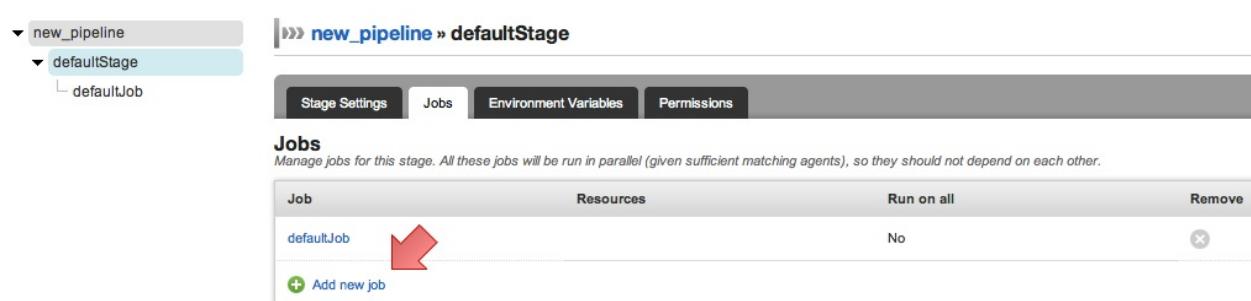
Now that we have a pipeline with stage(s), we can add more jobs to any of the existing stages. You can now use the tree navigation on the left side of the screen to edit a stage or a job under a pipeline.

- Click on the stage name that you want to edit on the tree as shown below. The "defaultStage" is being edited.



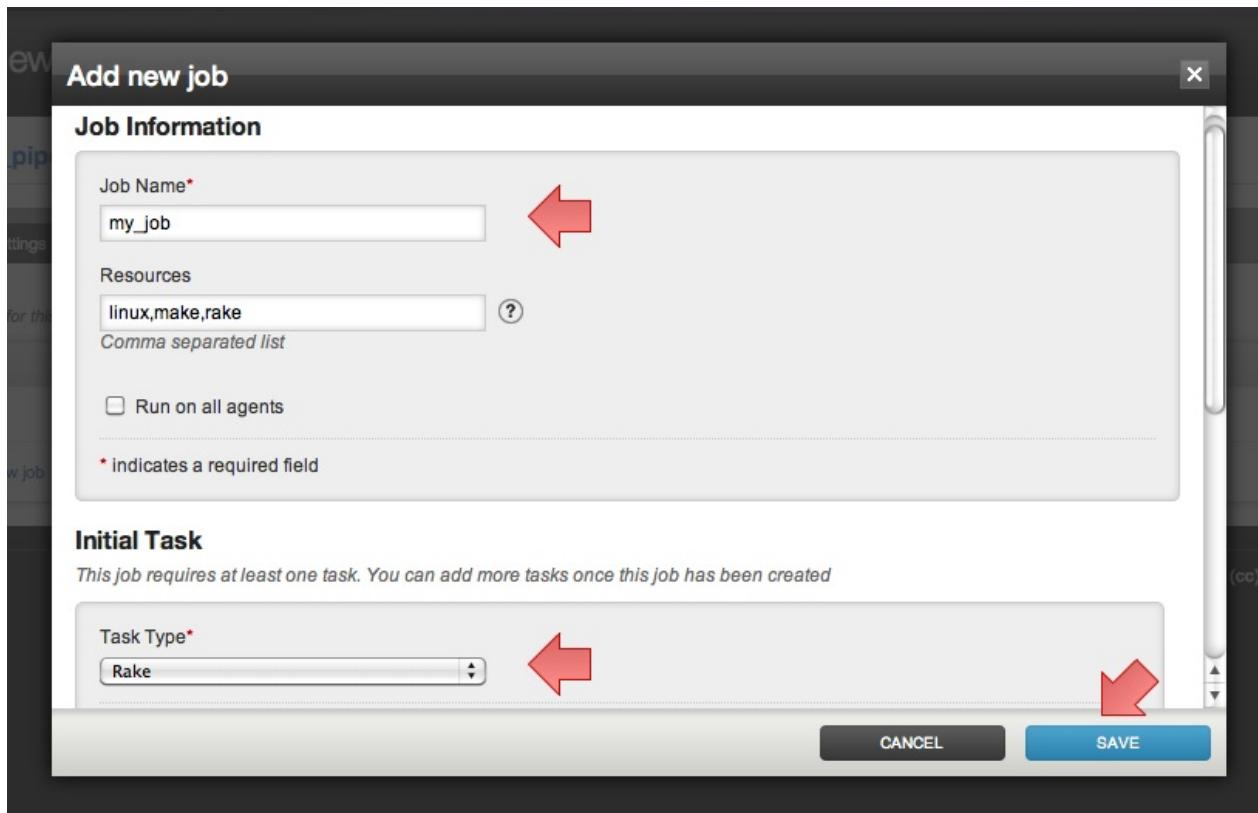
The screenshot shows the left sidebar of the application with a tree view. The 'new\_pipeline' node is expanded, revealing its children: 'defaultStage' and 'defaultJob'. The 'defaultStage' node is currently selected, indicated by a blue background. To the right of the tree, there is a detailed view of the 'defaultStage' configuration. At the top of this view is a navigation bar with tabs: General Options, Project Management, Materials, Stages, Environment Variables, and Parameters. The 'General Options' tab is active. Below the tabs, the 'Basic Settings' section is visible, containing a 'Pipeline Name\*' input field which is filled with 'new\_pipeline'. A red arrow points to the 'defaultStage' node in the tree view.

- Click on the Jobs tab
- Click on "Add new job"



The screenshot shows the 'Jobs' tab for the 'defaultStage' of the 'new\_pipeline'. The left sidebar still shows the tree view with 'new\_pipeline' expanded and 'defaultStage' selected. The main area has a header 'new\_pipeline » defaultStage'. Below the header is a navigation bar with tabs: Stage Settings, Jobs, Environment Variables, and Permissions. The 'Jobs' tab is active. Underneath the tabs, the heading 'Jobs' is followed by the sub-instruction: 'Manage jobs for this stage. All these jobs will be run in parallel (given sufficient matching agents), so they should not depend on each other.' A table lists the current job: 'defaultJob'. The table has columns: Job, Resources, Run on all, and Remove. A red arrow points to the 'Add new job' button at the bottom of the table.

- Fill job name and job details



- Fill in the details for the initial task belonging to this job. You can edit this job later to [add more tasks](#)
- You can choose the type of the task as required.
- For task types Ant, Nant and Rake, the build file and target will default as per the tool used. For example, Ant task, would look for build.xml in the working directory, and use the default task if nothing is mentioned.
- Click on help icon next to the fields to get additional details about the fields you are editing.
- Click "Save"

## Add a new task to an existing Job

Now that we have a pipeline with stage(s) containing job(s) we can add tasks to any of the existing jobs. You can now use the tree navigation on the left side of the screen to edit a job under a stage.

- Click on the job name that you want to edit on the tree as shown below. The "defaultJob" is being edited.

**Basic Settings**

Pipeline Name\*  
new\_pipeline

Automatic pipeline locking (?)

Label Template\*  
\${COUNT} (?)

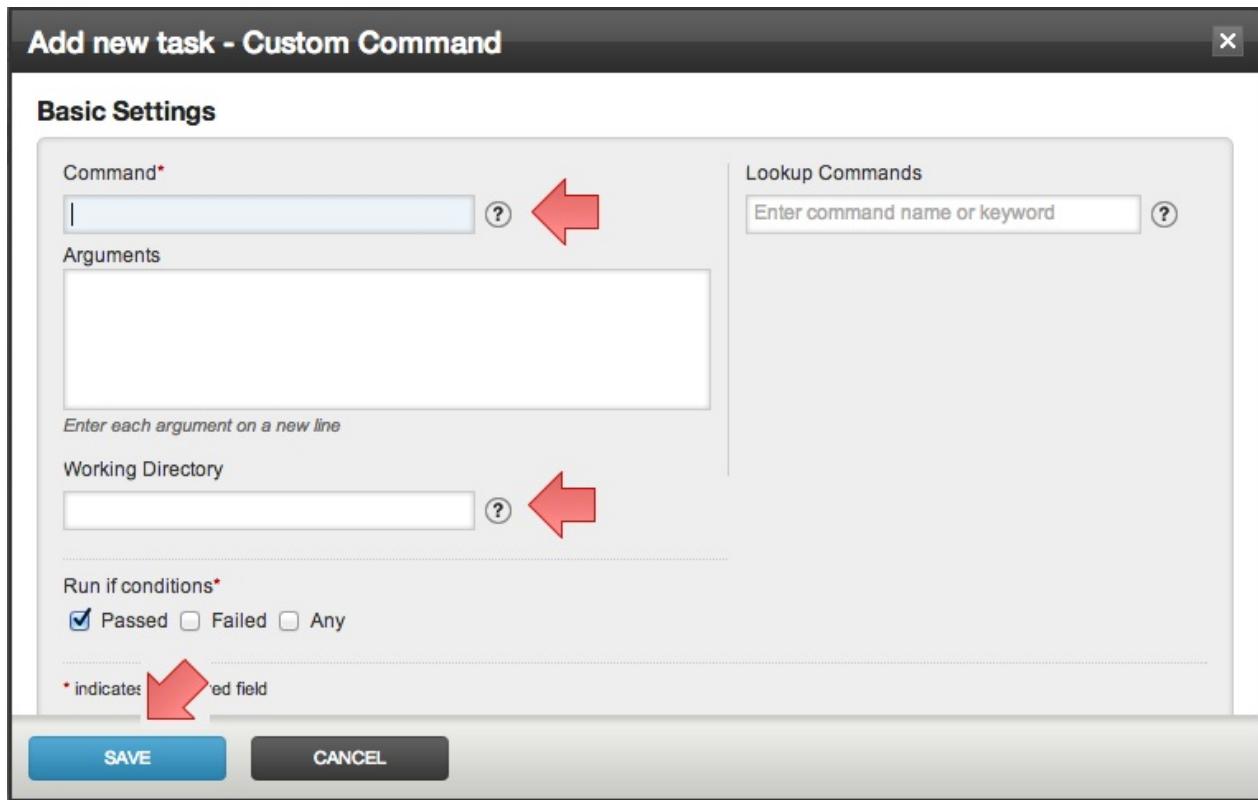
- Click on "Add new task". You can choose the task type from Ant, Nant, Rake and Fetch Artifact. Or you can choose "More..." to choose a command from [command repository](#) or specify your own command

**Tasks**

Order	Task Type	Run If Conditions	Properties	On Cancel	Remove
	Ant	Passed	Build File: build.xml	No	

+ Add new task ▾

- Fill the basic settings for the task
- Click on help icon next to the fields to get additional details about the fields you are editing.
- Click "Save"



- Advanced Options section allows you to specify a Task in which you can provide the actions (typically clean up) that needs to be taken when users chooses to cancel the stage.

## Clone an existing pipeline

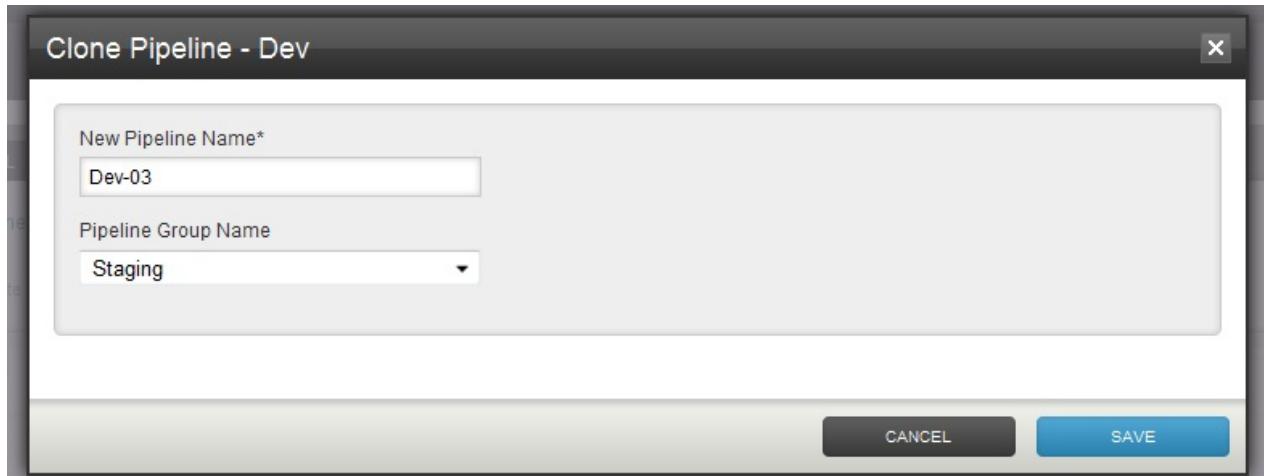
Clone pipeline functionality helps you create a new pipeline from an existing pipeline by giving it a new name. Typically when setting up a pipeline for a new branch, it is very useful to take an existing pipeline and clone it.

If the user is a pipeline group admin, she can clone the new pipeline into a group that she has access to. If the user is an admin she can clone the pipeline into any group or give a new group name, in which case the group gets created.

- Navigate to the Admin tab
- Locate the pipeline that needs to be cloned
- In that row, click on the "Clone" icon.

The screenshot shows a list of pipelines under a group named 'Staging'. The pipelines listed are 'Dev', 'Dev-01', and 'Dev-02'. Each pipeline has a set of actions: 'Edit', 'Move to', 'Clone' (highlighted by a red arrow), 'Delete', and 'Extract Template'. Below the list is a button to 'Create a new pipeline within this group'.

- Fill in the name of the new pipeline



- Select a pipeline group. If you are an admin, you will be able to enter the name of the pipeline group using the auto suggest or enter a new group name
- Click "Save"

## Pipeline Templates

Templating helps to create reusable workflows in order to make tasks like creating and maintaining branches, and managing large number of pipelines easier.

### Creating Pipeline Templates

Pipeline Templates can be managed from the Templates tab on the Administration Page.

**Pipeline Templates**

1. [+ Add New Template](#)

2. Pipeline

3. Delete

4. Edit

5. Pipeline

No pipelines associated with this template

**Add New Template**

**General Options**

Provides options to create a new template from scratch or from an existing pipeline

Template Name\*

Extract From Pipeline  app-trunk

\* indicates a required field

SAVE CANCEL

A template can also be extracted from a pipeline using the "Extract Template" link. This can be found on the "Pipelines" tab in the Administration page.

**Pipelines**

[+ Add New Pipeline Group](#)

my-app

Edit Delete

Actions
app-trunk

+ Create a new pipeline within this group

## Example

As an example, assume that there is a pipeline group called "my-app" and it contains a pipeline called "app-trunk" which builds the application from trunk. Now, if we need to create another pipeline called "app-1.0-branch" which builds 1.0 version of the application, we can use Pipeline Templates as follows

## Using Administration UI

- Create a template "my-app-build" by extracting it from the pipeline "app-trunk", as shown in the previous section.
- Create a new pipeline "app-1.0-branch" which defines SCM material with the branch url and uses the template "my-app-build".

## Using XML

### Editing Pipeline Templates

Go Administrators can now enable any Go user to edit a template by [making them a template administrator](#).

Template administrators can view and edit the templates to which they have permissions, on the template tab of the admin page. Template Administrators, will however not be able to add, delete or change permissions for a template. They will also be able to see the number of pipelines in which the template is being used, but not the details of those pipelines.

The screenshot shows the 'Pipeline Templates' page. At the top, there's a navigation bar with tabs for 'Templates' (which is active), 'Pipelines', and 'Jobs'. Below the navigation is a search bar with placeholder text 'Search for template or pipeline'. Underneath is a table header with columns 'Pipeline' and 'Actions'. A single row is visible in the table, representing 'template-1'. The row contains the pipeline name, a status icon, and three action buttons: 'Edit', 'Permissions', and 'Delete'. A note below the table states 'This template is used in 1 pipeline.'

## Viewing Pipeline Templates

Pipeline Templates can now be viewed by Administrators and Pipeline Group Administrators while editing or creating a Pipeline.

go PIPELINES ENVIRONMENTS AGENTS ADMIN

Pipelines > Service\_1 Paused by admin (Under construction)

Service\_1 Services-Template

**Service\_1**

General Options Project Management Materials Stages Environ

### Basic Settings

Pipeline Name\* Service\_1

Label Template\* \${COUNT} (?)

Automatic pipeline locking (?)

Automatic pipeline scheduling (?)  
Since this pipeline is based on a template, automatic/manual behaviour is determined by the template.

### Timer Settings

Cron Timer Specification (?)

Run only on new material (?)

Clicking on the icon indicated by arrow will display the following:

Services-Template

compile >> compile-job

Resources None 1

Job Timeout Use default

Run on all agents No

Tasks Artifacts Environment Variables Custom Tabs

go/service\_1\$ ant -f "build.xml" compile 2

On Cancel go/service\_1\$ ant -f "build.xml" clean 3

Run if Passed 4

conditionally, please ensure at least one of its materials has polling enabled.

The pop-up shows the extract of the template "Services-Template" configured for the

pipeline "Service\_1".

1. Shows the details of the job "compile-job" configured for the stage "compile".
2. Indicates that the working directory set for the task is "go/service\_1", which is followed by the "\$" symbol and then the command.
3. If any "On Cancel Task" has been configured, it will be indicated like this.
4. Shows the "Run If Condition" for this task.

## See also...

- [Templates - Configuration Reference](#)

## Stage approvals in action

---

By default, when one stage completes successfully, the next stage is automatically triggered by Go. However sometimes you don't want the next stage to be triggered automatically. This might be the case if you have a stage that deploys your application to a testing, staging or production environment. Another case can be when you don't want your pipeline to be automatically triggered by changes in version control. In these situations, you want the stage triggered by manual intervention. This can be done through manual [approvals](#).

If you add a manual approval to the first stage in a pipeline, it will prevent the pipeline from being triggered from version control. Instead, it will only pick up changes when you trigger the pipeline manually (this is sometimes known as "forcing the build").

From Cruise 1.1 (legacy version of Go), you can control who can trigger manual approvals. See the section on [Adding authorization to approvals](#) for more details.

## Managing pipeline groups

---

Starting with Cruise 1.3 (legacy version of Go), there is support for collecting multiple pipelines into a single named group. See the section on [Specifying who can view and operate pipeline groups](#) for more details.

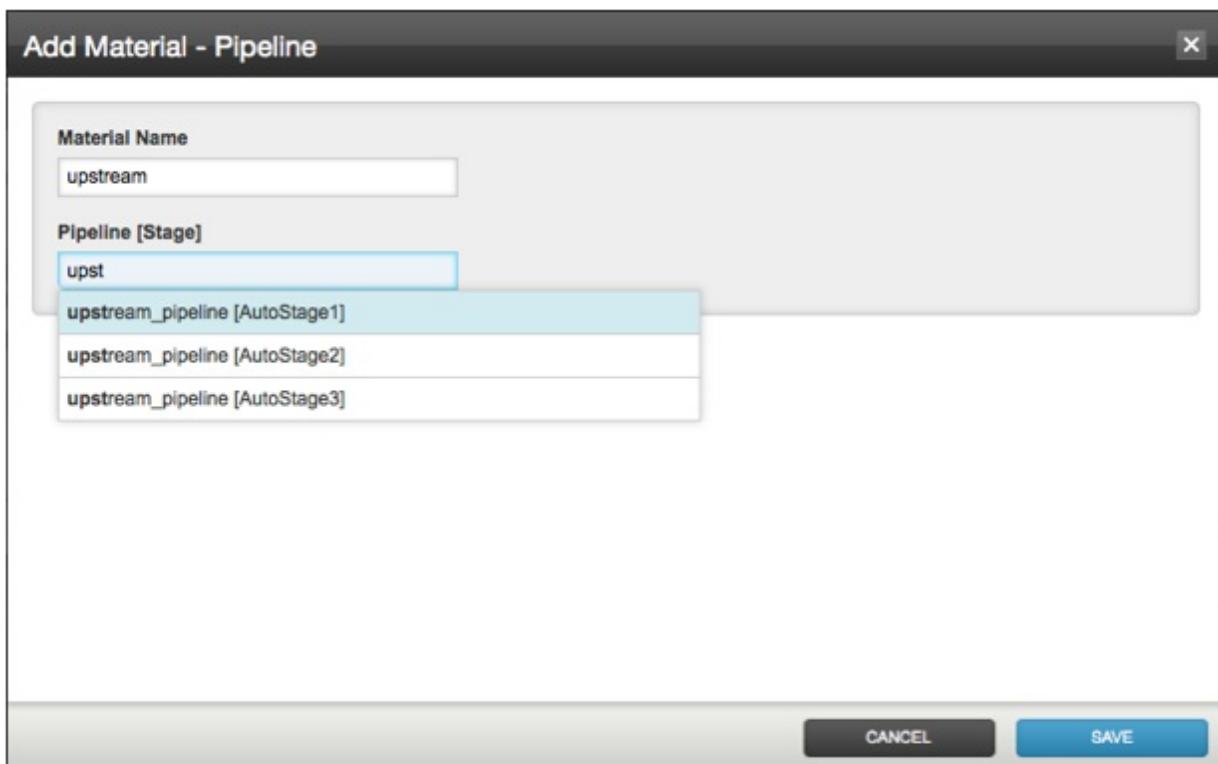
# Managing dependencies

Sometimes you need more complex triggers than a simple pipeline of stages and jobs. In particular, you may want a pipeline to trigger based on the result of a stage in another pipeline. This is possible by adding pipelines as materials.

## Creating a dependency

Say we have two pipelines - **upstream\_pipeline** and **downstream\_pipeline**. We want **downstream\_pipeline** to automatically trigger following the successful completion of the stage **AutoStage1** in pipeline **upstream\_pipeline**. Here's how we'd achieve this:

- Navigate to the **Admin** section.
- On the Pipelines screen, locate the **downstream\_pipeline** pipeline and **Edit** it.
- Click on the **Materials** tab.
- Add a new pipeline dependency material by clicking **Add Material** and then selecting **Pipeline**.
- You'll be presented with an **Add Material** popup (see screenshot)
- Enter **upstream\_pipeline [AutoStage1]** in the **Pipeline [stage]** field (it can also auto-complete)



Power users can also configure this via the **Config XML** tab on the Admin section:

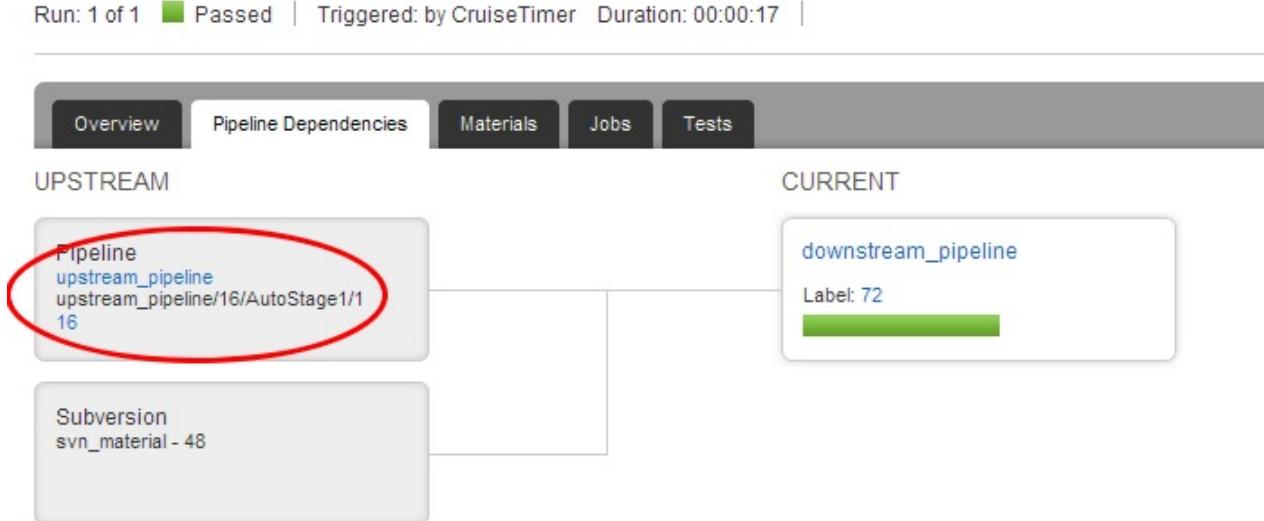
```
<pipeline name="downstream_pipeline">
  <materials>
    <pipeline pipelineName="upstream_pipeline" stageName="AutoStage1"/>
  </materials>
  ...
</pipeline>
```

Now, when the stage "AutoStage1" of "upstream\_pipeline" completes, the pipeline "downstream\_pipeline" will start building. The Pipeline Dependency visualization shows you all the downstream instances that were triggered off the upstream instance (label 14) currently being viewed.

The screenshot shows the Go CI interface with the following details:

- Header:** go
- Navigation:** PIPELINES, ENVIRONMENTS, AGENTS, ADMIN
- Pipeline View:** go-release-latest (Label: 13.1.0.182-f3eee89f304c368ab1e8bf0d4ccde16f87a70a2c) > dev
- Build Status:** dev, plugin-deps, dist, smoke-firefox, smoke-ie (all green)
- Run Details:** Run: 1 of 1 | Passed | Automatically triggered on 12 Feb, 2013 at 20:32:52 [+0530] | Duration: 00:40:10 | Compare
- Dependency Graph:** Overview, Pipeline Dependencies, Materials, Jobs, Tests, Config, Graphs
- UPSTREAM:** Git release-latest - f3eee89, Mercurial twist-release-latest - 8b2a5a21f058
- CURRENT:** go-release-latest (Label: 13.1.0.182-f3eee89f304...), showing 6 green status bars
- DOWNTREAM:** acceptance-release-latest (13.1.0.182-f3eee89f304...), 13.1.0.182-f3eee89f304..., Performance-Setup-Minor

If you want to view the materials that are associated with "downstream\_pipeline", the pipeline details page for that specific instance of the downstream pipeline will show you all this information.



## Fetching artifacts from an upstream pipeline

Go can automatically fetch artifacts from a previous stage of the current pipeline or from any ancestor pipeline it depends on. This is useful when a pipeline depends on binaries that are produced earlier in the pipeline.

Note that you can not specify two (or more) dependencies for the same upstream pipeline.

For example, in the following configuration, when the stage "AutoStage1" of pipeline "upstream\_pipeline" passes, the pipeline "downstream\_pipeline" starts, and the artifacts are fetched from the upstream pipeline in the stage 'Stage' of "downstream\_pipeline". You can see the exact pipeline and stage that triggered this in the sub-tab 'Materials' on the stage details page.

You can do this via the admin screens for the respective pipelines. You'll need to first define the artifact in the "upstream\_pipeline" at the job level:

Source	Destination	Type
target/commonlib.dll	pkg	Build Artifact
		Build Artifact

**Artifacts**

**Source**      **Destination**      **Type**

+ Add

**SAVE**      **RESET**

Then, you'll want to retrieve (fetch) that artifact from within the "downstream\_pipeline." You can do this by creating a "Fetch Artifact" task within a job in that pipeline. Since you have already defined "upstream\_pipeline" as a dependency material, artifacts from that pipeline are accessible in this pipeline.

**Add new task - Fetch Artifact**

**Basic Settings**

Pipeline: upstream\_pipeline

Stage: AutoStage1

Job: firstJob

Source: target/commonlib.dll

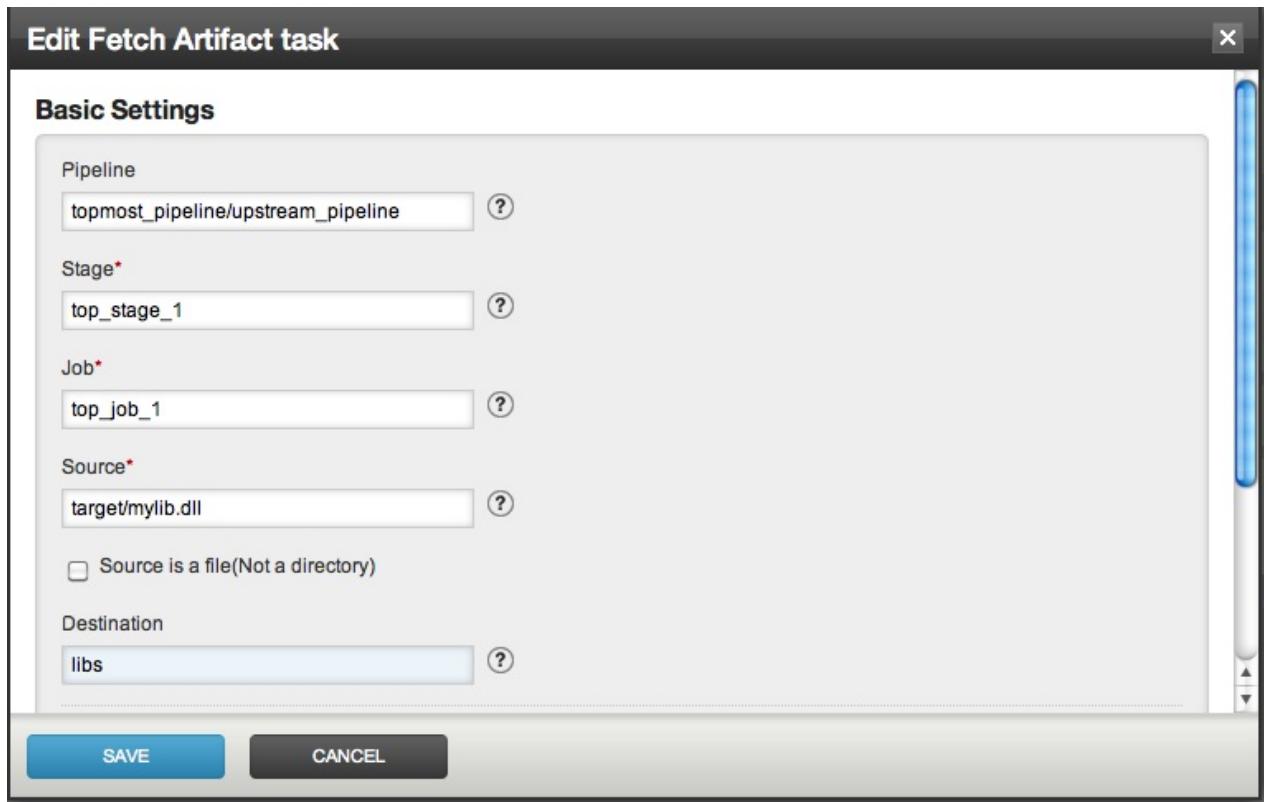
Source is a file (not a directory)

Destination:

**Run If Conditions**

**CANCEL**      **SAVE**

A fetch task can also be instructed to retrieve (fetch) an artifact from an ancestor pipeline. For example, lets assume that the "upstream\_pipeline" used in this example, depends on another pipeline "topmost\_pipeline". Then you can define a a "Fetch Artifact" task to fetch artifacts from "topmost\_pipeline" by defining the hierarchy of these pipelines as follows. You have to specify the hierarchy by separating the pipelines with a /. For example: topmost\_pipeline/upstream\_pipeline.



For power users, here's how you can configure this via Config XML:

```
<pipeline name="topmost_pipeline">
  <materials>
    <svn url="...."/>
  </materials>
  ...
  <stage name="TopStage1">
    <jobs>
      <job name="topJob">
        <tasks>
          <nant />
        </tasks>
        <artifacts>
          <artifact src="target/mylib.dll" dest="lib"/>
        </artifacts>
      </job>
    </jobs>
  </stage>
</pipeline>
<pipeline name="upstream_pipeline">
  <materials>
    <svn url="...."/>
    <pipeline pipelineName="topmost_pipeline" stageName="TopStage1"/>
  </materials>
  ...
  <stage name="AutoStage1">
    <jobs>
      <job name="firstJob">
        <tasks>
          <nant />
        </tasks>
        <artifacts>
```

```
        <artifact src="target/commonlib.dll" dest="pkg"/>
    </artifacts>
</job>
</jobs>
</stage>
</pipeline>
<pipeline name="downstream_pipeline">
<materials>
    <pipeline pipelineName="upstream_pipeline" stageName="AutoStage1"/>
</materials>
<stage name="Stage">
    <jobs>
        <job name="fetchFromParentJob">
            <tasks>
                <fetchartifact pipeline="upstream_pipeline" stage="AutoStage1" job="firstJob">
            </tasks>
        </job>
        <job name="fetchFromAncestorJob">
            <tasks>
                <fetchartifact pipeline="topmost_pipeline/upstream_pipeline" stage="TopStage1">
            </tasks>
        </job>
    <jobs>
    </stage>
    ...
</pipeline>
```

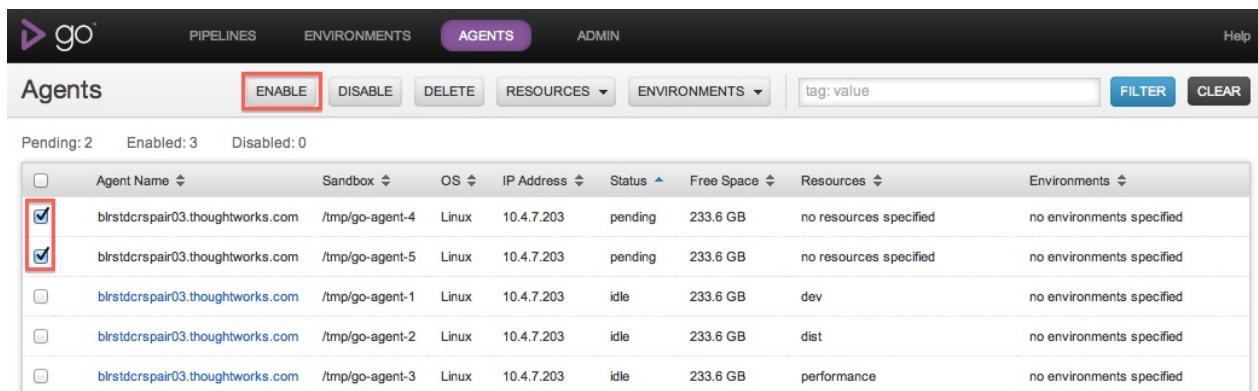
# Managing agents

Go is designed to make managing a build cloud extremely simple. This page takes you through the lifecycle of expanding your cloud and maintaining your agents.

## Adding a Go agent to your cloud

The first thing you need to do is [install Go agent](#) on the machine you want to add to the cloud.

Once the Go agent has been installed and pointed at your Go server, go to the [Agents](#) tab on the Go dashboard. You should see something like this:



A screenshot of the Go Agents dashboard. The top navigation bar includes links for PIPELINES, ENVIRONMENTS, AGENTS (which is highlighted in purple), and ADMIN. Below the navigation is a search bar with the placeholder "tag:value" and buttons for FILTER and CLEAR. A status summary shows Pending: 2, Enabled: 3, and Disabled: 0. The main table lists five agents, each with columns for Agent Name, Sandbox, OS, IP Address, Status, Free Space, Resources, and Environments. The first two agents in the list have their checkboxes checked and are highlighted with red boxes. The other three agents have their checkboxes unchecked.

Pending: 2	Enabled: 3	Disabled: 0						
<input type="checkbox"/>	Agent Name	Sandbox	OS	IP Address	Status	Free Space	Resources	Environments
<input checked="" type="checkbox"/>	birstdcrspair03.thoughtworks.com	/tmp/go-agent-4	Linux	10.4.7.203	pending	233.6 GB	no resources specified	no environments specified
<input checked="" type="checkbox"/>	birstdcrspair03.thoughtworks.com	/tmp/go-agent-5	Linux	10.4.7.203	pending	233.6 GB	no resources specified	no environments specified
<input type="checkbox"/>	birstdcrspair03.thoughtworks.com	/tmp/go-agent-1	Linux	10.4.7.203	idle	233.6 GB	dev	no environments specified
<input type="checkbox"/>	birstdcrspair03.thoughtworks.com	/tmp/go-agent-2	Linux	10.4.7.203	idle	233.6 GB	dist	no environments specified
<input type="checkbox"/>	birstdcrspair03.thoughtworks.com	/tmp/go-agent-3	Linux	10.4.7.203	idle	233.6 GB	performance	no environments specified

Note that the hostname is as reported by the agent, and the IP address is as reported by the server.

To add the agent to the cloud, click "Enable". Note that even after you have clicked "Enable", the agent will not be enabled until it next contacts the server -- so if your agent has stopped talking to the server, nothing will happen.

Once your agent has successfully been enabled, it should no longer appear greyed out and will and be marked "idle". At this point your agent will automatically begin picking up jobs. Agents will automatically check out any code they need in order to start running jobs.

## Matching jobs to agents

In its default state, Go server will assign scheduled jobs to the first available agent. Go

doesn't have the ability to determine what operating system or other resources are present on a given agent. If you want particular jobs to run on particular agents, you'll need to specify **resources**.

You can specify one or more resources that a particular job needs in order to execute. In the same way, you can specify that an agent has one or more resources. Go will then match jobs to agents, such that a job will only run on agents which have at least the resources required for that job.

Resources are just plain text tags. There are no preset tags or conventions around tagging -- just use what makes sense to you. You might, for example, use operating systems as tags: "RHEL linux", "Windows XP". You could also use browsers, databases, or whatever else makes sense. We recommend you let your classification be driven by your jobs -- if you know that certain jobs will only work on certain machines, tag the jobs with the special resource or resources that job needs in order to work, and then classify the agents accordingly.

Note: Resource matching is case-insensitive.

To specify the resources that a job needs, go to the **Pipelines configuration** section of the [Administration](#) tab and edit the job that you want to specify resources for:

The screenshot shows the 'Job Settings' tab selected in a navigation bar with tabs for 'Job Settings', 'Tasks', 'Artifacts', 'Environment Variables', and 'Custom Tabs'. The main area is titled 'Job Settings' and contains fields for 'Job Name\*' (with 'windows-1' entered) and 'Resources' (with 'windows,dev' entered). A note below says 'Comma separated list'. A question mark icon is next to the resources input field.

Once you've specified the resources your jobs need, you'll want to describe the resources your agents have. You can do this very easily in the [Agents](#) tab. Just select the agents you want to describe, and click on the **Resources** button.

## Associate selected agent(s) with a newly created resource

Enter the name of the new resource and click the “Add” button.

The screenshot shows the Go web interface with the Agents tab selected. A modal window is open for adding resources. The 'Create New' input field contains 'UAT02-twist-linux'. The 'ADD' button is highlighted with a red box.

## Associate selected agent(s) with existing resources

All existing resources across your agents and jobs will appear in alphabetical order. Select one of three states for all resources you want to associate and then click the “Apply” button.

- A resource **with a check** will add the resource to all selected agents.
- A resource **with a forward slash** means some of your selected agents are associated to it. No change will occur after clicking “Apply”.
- A resource **without a check** will remove the resource from all selected agents.

The screenshot shows the Go web interface with the Agents tab selected. A modal window is open for associating resources. The 'performance' checkbox is checked. The 'APPLY' button is highlighted with a red box.

## Agent states

Go will tell you if it loses touch with agents. If Go server doesn't hear from an agent for two minutes, the agent will turn red in the [Agents](#) tab, and Go will tell you the last time it heard from the agent. Go will also transparently re-assign the build to the next available agent that can run it, if the lost agent was building a job.

Go will also let you know if one of the agents it knows about has never contacted it since Go server was last started. In this case, the agent's state will be marked as "missing"

and it will be gray.

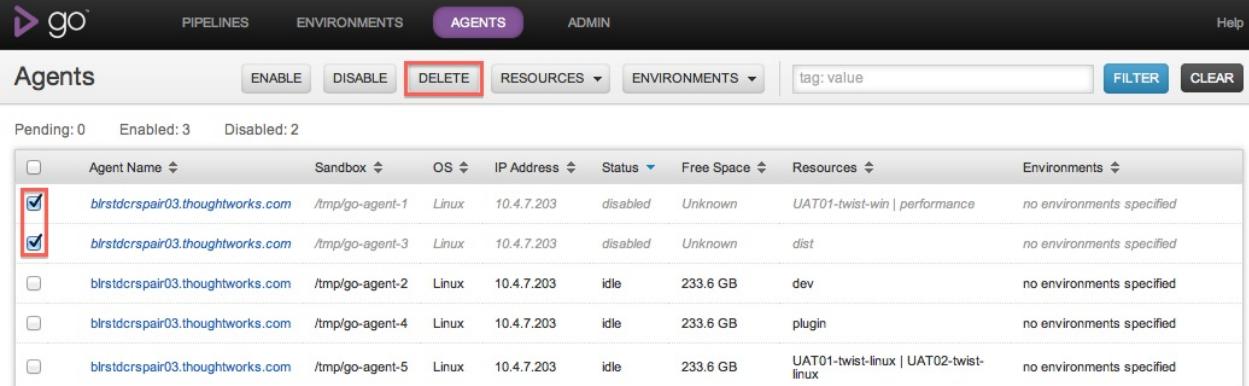
If an agent is working on a job, it will turn orange, and display the name of the job it is working on. You can click on the job description to go to the job details page for that job:

## Removing agents

If you want to remove an agent from Go's build cloud, go to the agents tab, locate the agent you want to remove, and click on the button marked "Disable". Go will record in its configuration that this agent should be excluded from the build cloud. If you restart Go server, the agent will continue to be disabled. Disabled agents do not count towards Go's licensed agents.

To permanently remove an agent from Go's configuration, you can use the [agent api](#) or delete from the agents tab. The agent must be disabled before it can be deleted

Following this procedure, if you restart the agent, Go server will see it as a new agent, and you can enable it again in the same way as described above.



Pending: 0	Enabled: 3	Disabled: 2						
<input type="checkbox"/>	Agent Name	Sandbox	OS	IP Address	Status	Free Space	Resources	Environments
<input checked="" type="checkbox"/>	blrstdcrspair03.thoughtworks.com	/tmp/go-agent-1	Linux	10.4.7.203	disabled	Unknown	UAT01-twist-win   performance	no environments specified
<input checked="" type="checkbox"/>	blrstdcrspair03.thoughtworks.com	/tmp/go-agent-3	Linux	10.4.7.203	disabled	Unknown	dist	no environments specified
<input type="checkbox"/>	blrstdcrspair03.thoughtworks.com	/tmp/go-agent-2	Linux	10.4.7.203	idle	233.6 GB	dev	no environments specified
<input type="checkbox"/>	blrstdcrspair03.thoughtworks.com	/tmp/go-agent-4	Linux	10.4.7.203	idle	233.6 GB	plugin	no environments specified
<input type="checkbox"/>	blrstdcrspair03.thoughtworks.com	/tmp/go-agent-5	Linux	10.4.7.203	idle	233.6 GB	UAT01-twist-linux   UAT02-twist-linux	no environments specified

## Pausing agents

If you want to pause an agent or temporarily disable it so that Go server will not assign work to the agent. Go will record in its configuration file that the agent has been disabled. This means, if you restart Go server, the disabled agent will remain disabled. You can use the following api to [disable agents](#) or you can disable the agent from the agents tab.

A disabled agent can be enabled; this will make it eligible to pick up work again. You can use the api or enable an agent from the agents tab.

Pending: 0	Enabled: 5	Disabled: 0					
Agent Name	Sandbox	OS	IP Address	Status	Free Space	Resources	Environments
<input checked="" type="checkbox"/> blrstdcrspair03.thoughtworks.com	/tmp/go-agent-5	Linux	10.4.7.203	lost contact	Unknown	UAT01-twist-linux   UAT02-twist-linux	no environments specified
<input checked="" type="checkbox"/> blrstdcrspair03.thoughtworks.com			10.4.7.203	missing	Unknown	UAT01-twist-win   performance	no environments specified
<input checked="" type="checkbox"/> blrstdcrspair03.thoughtworks.com	/tmp/go-agent-2	Linux	10.4.7.203	idle	233.6 GB	dev	no environments specified
<input type="checkbox"/> blrstdcrspair03.thoughtworks.com	/tmp/go-agent-3	Linux	10.4.7.203	idle	233.6 GB	dist	no environments specified
<input type="checkbox"/> blrstdcrspair03.thoughtworks.com	/tmp/go-agent-4	Linux	10.4.7.203	idle	233.6 GB	plugin	no environments specified

## Details of a single agent

Go now provides a page that shows the details of a single agent. This page provides details about the agent configuration and the history of all the jobs that ran on that agent.

### Agent Details tab

This tab shows the configuration and runtime information of an agent. For example, this tab shows the free space available on the agent, the IP Adress and the OS of the agent.

In terms of configuration, this tab shows the resources of the agent and the environment it belongs to. A sample Details tab looks as below:

Free Space:	Unknown
Sandbox:	C:\AutoDeployTesting\uat-agent
IP Address:	10.4.8.84
OS:	Windows 2003
Resources:	Windows2003   autodeploy   windows
Environments:	AutoDeploy-WindowsServer2003   sriram-spike-windows

### Job Run History tab

You must be logged in as an admin user to configure this step.

This tab shows a table of all the completed jobs that ran on this agent. A sample page is

shown below

The screenshot shows the go interface with the following navigation bar: PIPELINES, ENVIRONMENTS, AGENTS (highlighted in purple), and ADMIN. Below the navigation bar, the pipeline name 'blrstdcrsato04' is displayed. Underneath, there are two tabs: 'Details' (selected) and 'Job Run History'. The main area is a table with the following columns: Pipeline, Stage, Job, Result, and Completed. The table lists eight rows of data corresponding to the jobs shown in the screenshot.

Pipeline	Stage	Job	Result	Completed
sriram-windows-remote-cbgd	upgrade-go-server	upgrade	Passed	2013-02-11T10:36:42+05:30
sriram-windows-remote-cbgd	transferFile2node	robocopy	Passed	2013-02-11T10:33:01+05:30
sriram-windows-remote-cbgd	transferFile	fetch-go-binaries	Passed	2013-02-11T10:31:37+05:30
sriram-windows-remote-cbgd	upgrade-go-server	upgrade	Passed	2013-02-07T21:42:10+05:30
sriram-windows-remote-cbgd	upgrade-go-server	upgrade	Failed	2013-02-07T20:23:39+05:30
sriram-windows-remote-cbgd	transferFile2node	robocopy	Passed	2013-02-07T20:19:47+05:30
sriram-windows-remote-cbgd	transferFile	fetch-go-binaries	Passed	2013-02-07T20:18:17+05:30
sriram-windows-remote-cbgd	upgrade-go-server	upgrade	Passed	2013-02-07T16:21:02+05:30

For every job, the following columns are shown:

1. Pipeline: The pipeline to which the job belongs to
2. Stage: The stage to which the job belongs to
3. Job: The name of the job
4. Result: The result of the job - Passed, Failed, Cancelled or Rescheduled
5. Completed: The date when the Job completed
6. Duration: The duration that the Job took to finish - from scheduled till completed.

The job listing table can be sorted on any column, except for the Duration column.

## Using Agent details to debug agent issues

This page is useful to figure out if there are agent issues and hence a certain job keeps failing on that agent.

Consider a job which runs functional tests for a web application that need a browser to be available. The job was passing so far and only recently it has started to fail intermittently. Here are the steps you can follow to figure out if this is an agent issue.

1. Navigate to the [Job Details page](#) of the given job that failed.
2. Locate the "Agent" label and click on the link to the agent
3. Navigate to the "Job Run History" tab
4. Sort on the Job Name and locate the job that just navigated from

You'd notice that the job started to fail recently. You can even see if there are other jobs

that have started failing around the same time by now sorting on the Completed date.

# Managing environments

Go is configured using an XML configuration file. This file can be edited through the Go server dashboard. Go allows you to edit sections of the configuration independently and will check the syntax of the configuration before it saves it again. You can also edit the full XML file if you wish, by clicking on the Config XML section of the Administration tab.

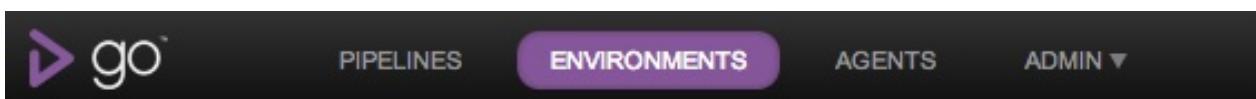
## Creating a new environment

An Environment is a grouping of pipelines and agents. By assigning an agent to an environment, it will be used to run only those jobs that belong to the pipelines of that environment. An agent can belong to more than one environment. This means, for instance, the same agent can be used to deploy something into an UAT or a Performance testing environment. A pipeline can, however, only be assigned to a single environment. Generally, these pipelines represent the tasks that need to happen in a given environment. For example deploying a 3-tier application into an UAT environment with 6 machines and running smoke tests on the setup.

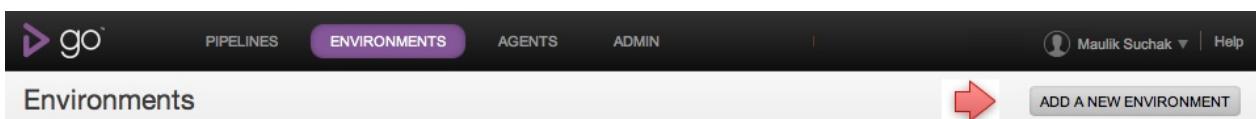
However, once an agent is associated with one or more environments, it is no longer capable of picking up jobs on pipelines that do not belong to environments. Pipelines outside of environments will only be assigned to agents in the default pool (not associated with any environment).

You can create an environment in the admin UI through the following steps. This example creates a production environment, adding the deployment pipeline and the agent installed on the production server.

- Click on the Environments tab



- Click on the "Add a new environment"



- Provide a name for the environment

## Add Environment

Step 1:  
Name

Step 2:  
Add Pipelines

Step 3:  
Add Agents

Step 4:  
Add Environment Variables

Environment Name:

Production

FINISH

NEXT

CANCEL

- Add one or more pipelines that need to run on the environment

## Add Environment

Step 1:  
Name

Step 2:  
Add Pipelines

Step 3:  
Add Agents

Step 4:  
Add Environment Variables

Pipelines to add:

Available pipelines

- Build
- Deploy-Prod
- Deploy-UAT

► Unavailable pipelines (Already associated with environments)

NEXT

FINISH

CANCEL

- Add one or more agents associated with the environment

## Add Environment

Step 1:  
Name

Step 2:  
Add Pipelines

Step 3:  
Add Agents

Step 4:  
Add Environment Variables

Agents to add:

<input type="checkbox"/>	Agent Name	Sandbox	OS	IP Address	Resources	Environments
<input type="checkbox"/>	Build-agent			20.4.31.61	mac	no environments specified
<input type="checkbox"/>	Build-agent2			20.41.4.23	mac	no environments specified
<input type="checkbox"/>	agent-on-UAT			20.10.15.15	mac	no environments specified
<input checked="" type="checkbox"/>	agent-on-prod			20.5.71.81	mac	no environments specified

NEXT

FINISH

CANCEL

- Add one or more environment variables that need to be passed.

Add Environment

Step 1: Name      Step 2: Add Pipelines      Step 3: Add Agents      Step 4: Add Environment Variables

Environment Variables (Name = Value)

version	=	10.0	X
type	=	PROD_01	X
<a href="#">+ Add</a>			

[FINISH](#)      [CANCEL](#)

- Click on finish

Setting up an environment through the xml can be found in the [configuration reference](#)

## Add a new agent to an existing environment

You can do this very easily in the Agents tab. Just select the agents you want to add to your environment and click on the Environments button. All existing environments will appear in alphabetical order. Select one of three states for all environments you want to add and then click the “Apply” button.

- An environment **with a check** will add the environment to all selected agents.
- An environment **with a forward slash** means some of your selected agents are associated to it. No change will occur after clicking “Apply”.
- An environment **without a check** will remove the environment from all selected agents.

go

PIPLINES ENVIRONMENTS AGENTS ADMIN

Agents

Pending: 0	Enabled: 56	Disabled: 20			
<input type="checkbox"/>	Agent Name	Sandbox	OS	IP Ad	
<input checked="" type="checkbox"/>	birstdcrsato04	C:\AutoDeployTesting\uat-agent	Windows 2003	10.4.8.82	
<input checked="" type="checkbox"/>	birstdcrsato06.thoughtworks.com	/export/users/cruise_builder/Auto DeployTesting/go-agent-12.4.0	Linux	10.4.8.83	
<input checked="" type="checkbox"/>	birstdgoato09	/export/users/cruise_builder/Auto DeployTesting/uat-agent	Linux	10.4.8.84	
<input checked="" type="checkbox"/>	birstdgoato02			10.4.8.82	missing Unknown
<input type="checkbox"/>	birstdgoato15			10.4.8.93	missing Unknown

ENVIRONMENTS tag: value [FILTER](#) [CLEAR](#)

AutoDeploy-CentOS\_5.4  
 AutoDeploy-FedoraCore  
 AutoDeploy-Ubuntu10.04  
 AutoDeploy-Ubuntu\_12.04  
 AutoDeploy-Windows7  
 AutoDeploy-WindowsServer2003  
 AutoDeploy-WindowsServer2008  
 Development

[APPLY](#)

Resources	Environments
Windows2003   autodeploy   windows	AutoDeploy-WindowsServer2003   siram-spike-windows
CentOS_5.4   autodeploy   linux	AutoDeploy-CentOS_5.4
Ubuntu_12.04   autodeploy	AutoDeploy-Ubuntu_12.04
Windows7   autodeploy   windows	AutoDeploy-Windows7
Windows2008   autodeploy   windows	AutoDeploy-WindowsServer2008

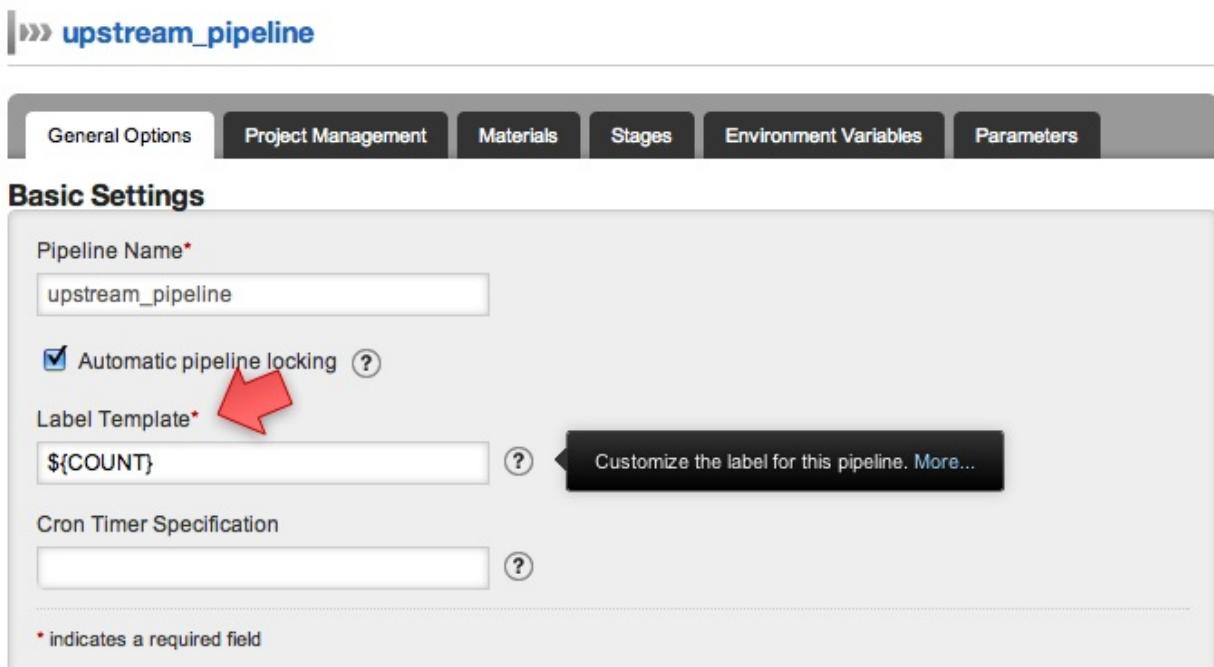
# Pipeline Labelling

Go maintains an internal counter to identify a pipeline. This number increases by 1 for each build. By default, Go will use this counter as the pipeline label. This label is also passed to your build as an environment variable: **GO\_PIPELINE\_COUNTER**. The pipeline counter increases even if a build fails.

The concept of pipeline counters was introduced in release 1.3.2. In order to maintain backward compatibility with historical data Go now uses negative values as counter for pipelines created by older releases of Go. Hence it is perfectly normal for a historical pipeline to have a negative counter with positive label.

## Customising the pipeline label

You can create a custom label by setting the **Label Template** field on your pipeline. This will change the value that Go shows on its webpages. It will also change the value of the **GO\_PIPELINE\_LABEL** property that is passed to your build. You can refer to  `${COUNT}` or material names which are defined in the configuration of [materials](#).



The screenshot shows the 'Basic Settings' page for a pipeline named 'upstream\_pipeline'. At the top, there is a navigation bar with tabs: General Options, Project Management, Materials, Stages, Environment Variables, and Parameters. The 'General Options' tab is selected. Below the tabs, the 'Basic Settings' section is visible. The 'Pipeline Name\*' field contains 'upstream\_pipeline'. The 'Automatic pipeline locking' checkbox is checked. The 'Label Template\*' field contains the placeholder text '\${COUNT}'. A red arrow points from the text above this image to the 'Label Template' field. To the right of the 'Label Template' field is a button labeled 'Customize the label for this pipeline. More...'. At the bottom of the form, there is a note: '\* indicates a required field'.

Power users can still edit the config xml to achieve the same. The xml snippet to configure **labelTempalte** is below.

```
<pipeline name="my-pipeline" labeltemplate="1.2.${COUNT}">
  ...
</pipeline>
```

### Using a **pipeline** in the labeltemplate:

```
<pipeline name="my-dependent-pipeline" labeltemplate="${MY_PIPELINE}">
  <materials>
    <pipeline pipelineName="my-pipeline" stageName="my-stage"/>
  </materials>
  ...
</pipeline>
```

### Using a **VCS material** in the labeltemplate. In this example, the Subversion revision number will be used as the labeltemplate:

```
<pipeline name="my-material-pipeline" labeltemplate="1.2.${SVN_MATERIAL}">
  <materials>
    <svn url="http://svn.example.com/" dest="svn" materialName="SVN_MATERIAL">
  </materials>
  ...
</pipeline>
```

# Pipeline Scheduling

Pipelines get scheduled automatically by default. Please see the knowledge base article in the Also see section below. Here we'll see how to disable automatic scheduling.

» demo

General Options Project Management Materials Stages Environment Variables Parameters

## Basic Settings

Pipeline Name\*

Label Template\*

Automatic pipeline locking

(?)

Automatic pipeline scheduling

(?)

If unchecked, this pipeline will only schedule in response to a Manual/API/Timer trigger. Unchecking this box is the same as making the first stage manual.

## Disable automatic scheduling

Unchecking the "Automatic Pipeline Scheduling" checkbox above disables auto scheduling. Actually this is the same as marking first stage as manual. We have just surfaced the option at a pipeline level to make it easier to spot. Please note though that this isn't really a pipeline level configuration. For example, if this is pipeline is based off a template, the checkbox above will be grayed out to indicate that it can only be toggled by editing the first stage in the template.

## Also see...

- [Different Types of Triggers for a Pipeline](#)

# Parameterize your Configuration

Go allows you to parameterize your pipelines and pipeline templates. This powerful feature can help reduce repetition within your configurations and also allows for complex setups using a combination of parameters and pipeline templates.

## Using Web Interface

Edit the **Parameters** tab when defining a **pipeline**.

Name	Value
ENV	= prod
MYSQL_VERSION	= 5.5

+ Add

## Defining and using a parameter

Parameter values are defined in the `<params>` tag within a pipeline and can be used anywhere within that pipeline using `#{param_name}`. The following example defines a new parameter called "myParam" and uses it in a job.

```
<pipeline name="my_pipeline">
  <params>
    <param name="myParam">hello world</param>
  </params>
  <stage name="my_stage">
    <jobs>
      <job name="my_job">
        <tasks>
          <exec command="echo" args="#{myParam}" />
        </tasks>
      </job>
    </jobs>
  </stage>
</pipeline>
```

**NOTE:** If you want to use the # literal, you can escape it using another # literal.

For example, if the parameter "foo" has the value "one", then:

String	Evaluates to
<code>#{foo}</code>	one
<code>##{foo}</code>	<code>#{foo}</code>
<code>###{foo}</code>	<code>#one</code>

## Using parameters in pipeline templates

Parameter usage within [templates](#) is similar to usage within pipelines. The only difference being, you cannot define parameters in a template.

```
<pipeline name="trunk" template="my_template">
  <params>
    <param name="WORKING_DIR">trunk</param>
  </params>
  ...
</pipeline>

<pipeline name="branch" template="my_template">
  <params>
    <param name="WORKING_DIR">branch</param>
  </params>
  ...
</pipeline>
```

The parameter defined above is used the template below.

```
<pipeline name="my_template">
  <stage name="my_stage">
    <jobs>
      <job name="my_job">
        <tasks>
          <exec command="echo" args="Updating code from svn repository svn://codebase">
        </tasks>
      </job>
    </jobs>
  </stage>
</pipeline>
```

# Rules around usage of parameters

---

While parameters are generally very flexible, there are some restrictions.

## You cannot use a parameter to define:

- Pipeline name
- Stage name
- Job name
- A Job's property name
- The \ configuration for a job's task
- Another parameter (i.e. you cannot define a parameter using another parameter)
- Pipeline template name
- Material name
- Material passwords (however, for Git and Mercurial, passwords are not captured as separate attribute, hence can be parameterized)
- Trigger-type for Stage

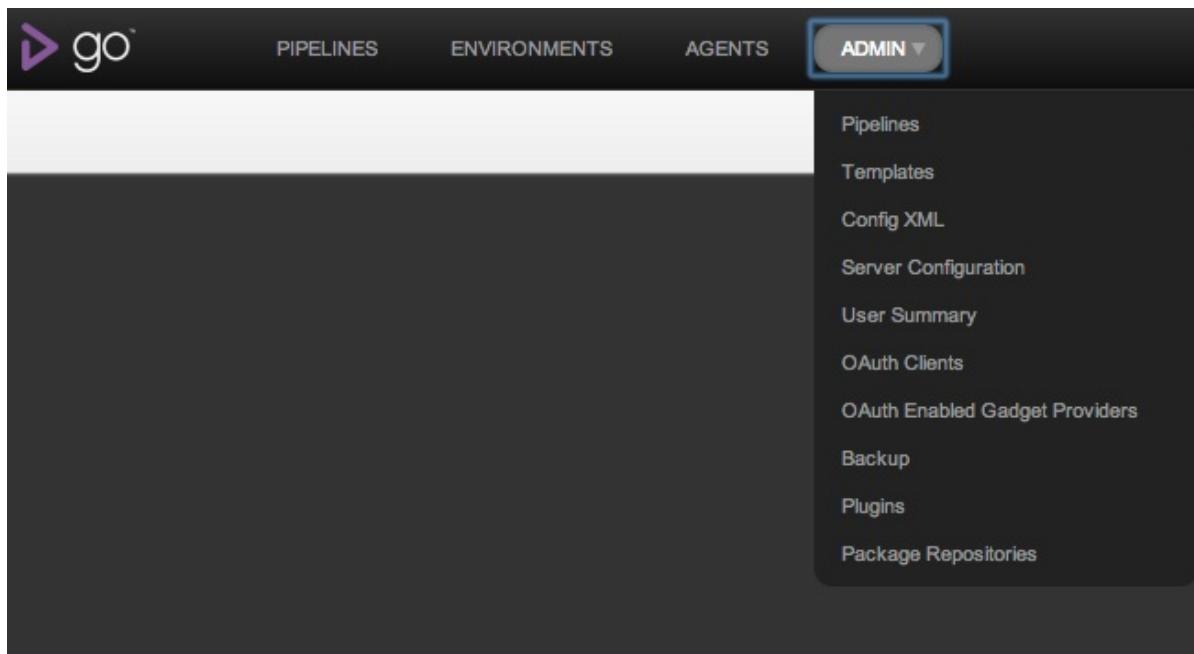
## Other restrictions:

- Parameters can currently only be defined within a pipeline.
- A parameter cannot be composed with another parameter i.e. #{foo#{bar}} will not be evaluated recursively.
- If a parameter is referenced but is not defined, then the configuration is invalid (Go will not let you save an invalid configuration).

# Use a custom pipeline label

When using Go to build your application, it is often useful to be able to include extra information in the label Go uses. For example, you might want to have your label contain a static major.minor version number in addition to the unique count of the pipeline.

- Click on the [Administration](#) tab



- Edit the pipeline



- Add the label template

General Options

Project Management

Materials

Stages

Environment Variables

Parameters

## Basic Settings

Pipeline Name\*

cruise

 Automatic pipeline locking [?](#)

Label Template\*

1.3.0.\${COUNT}

[?](#)

Cron Timer Specification

[?](#)

\* indicates a required field

- Click save

You might also want to include material revision into the pipeline label so that it's easier to find a Go pipeline by material revision and vice versa. For example, you might have a pipeline with a svn material. The following example shows how to include svn material revision into pipeline label:

```
<pipeline name="main" labeltemplate="1.3.${COUNT}-${svn}">
  <materials>
    <svn url="http://server/path" materialName="svn" />
  <materials>
  ...
</pipeline>
```

You can also include the revision of an upstream pipeline into the pipeline label to, for example, share the same revision across different but related pipelines:

```
<pipeline name="upstream" labeltemplate="1.3.${COUNT}-${svn}">
  <materials>
    <svn url="http://server/path" materialName="svn" />
  <materials>
  ...
</pipeline>
<pipeline name="downstream" labeltemplate="${upstream}">
  <materials>
    <pipeline pipelineName="upstream" stageName="dev" materialName="upstream" />
  <materials>
  ...
</pipeline>
```

In this case, if the label of upstream pipeline is "1.3.0-1234", then when downstream pipeline is triggered, the label of downstream pipeline is also "1.3.0-1234".

# Clone an existing pipeline

Clone pipeline functionality helps you create a new pipeline from an existing pipeline by giving it a new name. Typically when setting up a pipeline for a new branch, it is very useful to take an existing pipeline and clone it.

If the user is a pipeline group admin, she can clone the new pipeline into a group that she has access to. If the user is an admin she can clone the pipeline into any group or give a new group name, in which case the group gets created.

- Navigate to the Admin tab
- Locate the pipeline that needs to be cloned
- In that row, click on the "Clone" icon.

The screenshot shows a list of pipelines in a 'Staging' group. The pipelines listed are 'Dev', 'Dev-01', and 'Dev-02'. Each pipeline row has an 'Actions' column containing buttons for 'Edit', 'Move to', 'Clone' (highlighted with a red arrow), 'Delete', and 'Extract Template'. Below the table, there is a link '+ Create a new pipeline within this group'.

- Fill in the name of the new pipeline

The dialog box is titled 'Clone Pipeline - Dev'. It contains two input fields: 'New Pipeline Name\*' with the value 'Dev-03' and 'Pipeline Group Name' with the value 'Staging'. At the bottom of the dialog are 'CANCEL' and 'SAVE' buttons.

- Select a pipeline group. If you are an admin, you will be able to enter the name of the pipeline group using the auto suggest or enter a new group name
- Click "Save"

# Ensure only one instance of a pipeline can run at the same time

Sometimes you want to ensure that only a single instance of a pipeline can run at a time. This is important if the stages of a pipeline are interrelated. For example the first stage may set up an environment that is used by the next stage in the pipeline.

If a pipeline is locked then Go will not allow any other instance of that pipeline to be scheduled until the currently running one has been completed.

To enable locking from the Config UI, navigate to the General Options section of pipeline.

» demo

General Options Project Management Materials Stages Environment Variables Parameters

## Basic Settings

Pipeline Name\*

demo

Label Template\*

`\${COUNT}

Automatic pipeline locking ?

Automatic pipeline scheduling ?

If checked, only a single instance of the pipeline can be run at a time and the lock will release only if the entire pipeline completes successfully. This is particularly useful in deployment scenarios.

To enable locking from the Config XML set the `isLocked` attribute to true

```
<pipeline name="my-locked-pipeline" isLocked="true" >
  <materials>
    ...
  </materials>
  <stages>
    ...
  </stages>
</pipeline>
```

Also see the [configuration reference](#).

# Add a new material to an existing pipeline

Now that you have a pipeline, lets add another material to it.

- Navigate to the new pipeline you created by clicking on the **Edit** link under the Actions against it. You can also click on the name of the pipeline.

Pipelines [+ Add New Pipeline Group](#)

**my\_group** [Edit](#) [Delete](#)

Pipeline	Actions
new_pipeline	<a href="#">Edit</a> <a href="#">Move to ▾</a> <a href="#">Clone</a> <a href="#">Delete</a> <a href="#">Extract Template</a>

- Click on the Materials tab.

**new\_pipeline**

General Options Project Management Materials Stages Environment Variables Parameters

### Basic Settings

Pipeline Name\*  
new\_pipeline

Label Template\*  
\${COUNT} [?](#)

Automatic pipeline locking [?](#)  
 Automatic pipeline scheduling [?](#)

- You will notice an existing material . Click on the "Add new material" link.

**new\_pipeline**

General Options Project Management Materials Stages Environment Variables Parameters

### Materials

Name	Type	URL	Remove
https://svn-repo/svn/test/	Subversion	https://svn-repo/svn/test/	<a href="#">X</a>

[+ Add Material ▾](#)

- You will get the following message

**Add Material - Subversion**

**⚠** In order to configure multiple materials for this pipeline, each of its material needs have to a 'Destination Directory' specified. Please edit the existing material and specify a 'Destination Directory' in order to proceed with this operation.

**CLOSE**

- Edit the existing material and specify the destination directory

## Edit Material - Subversion



Material Name

 (?)

URL\*

Poll for new changes

Username

Password

Change Password

Check Externals

Destination Directory

**SAVE**

**CANCEL**

- Click "Save".

## Blacklist

Often you do want to specify a set of files that Go should ignore when it checks for changes. Repository changesets which contain only these files will not automatically trigger a pipeline. These are detailed in the [ignore](#) section of the [configuration reference](#).

- Enter the items to blacklist using ant-style syntax below

## Edit Material - Subversion



Password

Change Password

Check Externals

Destination Directory

**CHECK CONNECTION**

\* indicates a required field

**Blacklist**



Enter the paths to be excluded. Separate multiple entries with a comma.

**SAVE**

**CANCEL**

- Click "Save".

# Add a new stage to an existing pipeline

Now that you have a pipeline with a single stage, lets add more stages to it.

- Navigate to the new pipeline you created by clicking on the **Edit** link under the Actions against it. You can also click on the name of the pipeline.

Pipelines + Add New Pipeline Group

my\_group Edit Delete

Pipeline	Actions
new_pipeline	<span>Edit</span> <span>Move to</span> <span>Clone</span> <span>Delete</span> <span>Extract Template</span>

A red arrow points to the pipeline name 'new\_pipeline' in the list, and another red arrow points to the 'Edit' button in the actions column.

- Click on the Stages tab.

new\_pipeline

General Options Project Management Materials Stages Environment Variables Parameters

### Basic Settings

Pipeline Name\*  
new\_pipeline

Label Template\*  
\${COUNT} (?)

Automatic pipeline locking (?)  
 Automatic pipeline scheduling (?)

A red arrow points to the 'Stages' tab in the navigation bar, and another red arrow points to the 'Add new stage' link at the bottom of the stages table.

- You will notice that a defaultStage exists. Click on the "Add new stage" link.

new\_pipeline

General Options Project Management Materials Stages Environment Variables Parameters

### Stages

Configuration Type  Define Stages  Use Template

Order	Stage	Trigger Type	Jobs	Remove
	defaultStage	On Success	1	<span>(X)</span>
	<span>+ Add new stage</span>			

A red arrow points to the 'Add new stage' link at the bottom of the stages table.

- Fill stage name and trigger type.
- Fill in the details for the first job and first task belonging to this job. You can [add more jobs](#) and [add more tasks](#) to the jobs.
- Click on help icon next to the fields to get additional details about the fields you are editing.

## Stage Information

Stage Name\*

Trigger Type: 

On Success

Manual

\* indicates a required field

## Initial Job and Task

You can add more jobs and tasks to this stage once the stage has been created.

Job Name\*

- Click "Save".

# Add a new job to an existing stage

Now that we have a pipeline with stage(s), we can add more jobs to any of the existing stages. You can now use the tree navigation on the left side of the screen to edit a stage or a job under a pipeline.

- Click on the stage name that you want to edit on the tree as shown below. The "defaultStage" is being edited.

The screenshot shows the 'Basic Settings' section of the pipeline editor. The pipeline name is 'new\_pipeline'. The 'Stages' tab is selected. The tree on the left shows 'new\_pipeline' expanded, with 'defaultStage' selected. A red arrow points to the 'defaultStage' node in the tree.

- Click on the Jobs tab
- Click on "Add new job"

The screenshot shows the 'Jobs' section of the stage settings. The stage name is 'defaultStage'. The 'Jobs' tab is selected. The table lists one job: 'defaultJob'. A red arrow points to the '+ Add new job' button at the bottom of the table.

- Fill job name and job details

The screenshot shows the 'Add new job' dialog box. The 'Job Information' section includes fields for 'Job Name\*' (set to 'my\_job') and 'Resources' (set to 'linux,make,rake'). The 'Initial Task' section includes a 'Task Type\*' dropdown set to 'Rake'. A red arrow points to the 'Job Name' field. Another red arrow points to the 'Task Type' dropdown. A third red arrow points to the 'SAVE' button at the bottom right of the dialog.

- Fill in the details for the initial task belonging to this job. You can edit this job later to

### [add more tasks](#)

- You can choose the type of the task as required.
- For task types Ant, Nant and Rake, the build file and target will default as per the tool used. For example, Ant task, would look for build.xml in the working directory, and use the default task if nothing is mentioned.
- Click on help icon next to the fields to get additional details about the fields you are editing.
- Click "Save"

# Add a new task to an existing Job

Now that we have a pipeline with stage(s) containing job(s) we can add tasks to any of the existing jobs. You can now use the tree navigation on the left side of the screen to edit a job under a stage.

- Click on the job name that you want to edit on the tree as shown below. The "defaultJob" is being edited.

The screenshot shows the 'Basic Settings' section for the 'defaultJob' under the 'new\_pipeline' pipeline. The tree navigation on the left shows 'new\_pipeline' expanded, with 'defaultStage' and 'defaultJob' listed. A red arrow points to the 'defaultJob' node. The main panel contains fields for 'Pipeline Name' (set to 'new\_pipeline'), 'Automatic pipeline locking' (unchecked), and 'Label Template' (set to '\${COUNT}').

- Click on "Add new task". You can choose the task type from Ant, Nant, Rake and Fetch Artifact. Or you can choose "More..." to choose a command from [command repository](#) or specify your own command

The screenshot shows the 'Tasks' section for the 'defaultJob'. It lists one task: an 'Ant' task named 'Passed' with 'Build File: build.xml'. Below the table is a green button labeled '+ Add new task' with a red arrow pointing to it.

- Fill the basic settings for the task
- Click on help icon next to the fields to get additional details about the fields you are editing.
- Click "Save"

## Add new task - Custom Command

### Basic Settings

Command\*



Lookup Commands



Arguments

Enter each argument on a new line

Working Directory



Run if conditions\*

Passed  Failed  Any

\* indicates a required field

SAVE

CANCEL

- Advanced Options section allows you to specify a Task in which you can provide the actions (typically clean up) that needs to be taken when users chooses to cancel the stage.

## Add new task - Custom Command

Run if conditions\*

Passed  Failed  Any

\* indicates a required field

### Advanced Options

On Cancel Task

Ant



Build file



Target



Working directory



SAVE

CANCEL

# Pipeline Templates

Templating helps to create reusable workflows in order to make tasks like creating and maintaining branches, and managing large number of pipelines easier.

## Creating Pipeline Templates

Pipeline Templates can be managed from the Templates tab on the Administration Page.

The screenshot shows the Pipeline Templates section of the administration interface. At the top, there's a navigation bar with tabs: Pipelines, Templates (which is selected), Config XML, Server Configuration, User Summary, OAuth Clients, OAuth Enabled Gadget Providers, and Back. Below the navigation bar, the title "Pipeline Templates" is displayed, followed by a green "Add New Template" button with a plus sign. A red arrow labeled "1" points to this button. The main content area lists two templates: "template-pipeline" and "unused". The "template-pipeline" template is associated with a "prod\_pipeline" pipeline, which is listed under the "Actions" column. Red arrows labeled "2" and "5" point to the "Edit" and "Permissions" buttons for this pipeline respectively. Another red arrow labeled "4" points to the "Edit" button for the "unused" template. A message at the bottom states "No pipelines associated with this template".

Clicking on the "Add New Template" brings up the following form which allows you to create a fresh template, or extract it from an existing pipeline. Once saved, the pipeline indicated will also start using this newly created template.

**Add New Template**

**General Options**  
Provides options to create a new template from scratch or from an existing pipeline

Template Name\*

Extract From Pipeline [?](#)  
 ▾

\* indicates a required field

**SAVE** **CANCEL**

A template can also be extracted from a pipeline using the "Extract Template" link. This can be found on the "Pipelines" tab in the Administration page.

Pipelines    Templates    Source XML    Server Configuration    User Summary    User Roles    OAuth Clients    OAuth Enabled Gadget Providers    Backup

Pipelines    [+ Add New Pipeline Group](#)

my-app    [Edit](#)    [Delete](#)

Pipeline	Actions
app-trunk	<a href="#">Edit</a> <a href="#">Clone</a> <a href="#">Delete</a> <a href="#">+ Extract Template</a>

[+ Create a new pipeline within this group](#)

## Example

As an example, assume that there is a pipeline group called "my-app" and it contains a pipeline called "app-trunk" which builds the application from trunk. Now, if we need to create another pipeline called "app-1.0-branch" which builds 1.0 version of the application, we can use Pipeline Templates as follows

## Using Administration UI

- Create a template "my-app-build" by extracting it from the pipeline "app-trunk", as shown in the previous section.
- Create a new pipeline "app-1.0-branch" which defines SCM material with the branch url and uses the template "my-app-build".

# Using XML

## Editing Pipeline Templates

Go Administrators can now enable any Go user to edit a template by [making them a template administrator](#).

Template administrators can view and edit the templates to which they have permissions, on the template tab of the admin page. Template Administrators, will however not be able to add, delete or change permissions for a template. They will also be able to see the number of pipelines in which the template is being used, but not the details of those pipelines.

The screenshot shows a web-based administration interface for pipeline templates. At the top, there's a navigation bar with a 'Templates' tab selected. Below it, the main title is 'Pipeline Templates' with a 'Add New Template' button. A single template is listed: 'template-1'. For this template, there are 'Edit', 'Permissions', and 'Delete' buttons. A message below the template says 'This template is used in 1 pipeline.' There's also a header row with 'Pipeline' and 'Actions' columns.

## Viewing Pipeline Templates

Pipeline Templates can now be viewed by Administrators and Pipeline Group Administrators while editing or creating a Pipeline.

go PIPELINES ENVIRONMENTS AGENTS ADMIN

Pipelines > Service\_1 Paused by admin (Under construction)

Service\_1 Services-Template

**Service\_1**

General Options Project Management Materials Stages Environ

### Basic Settings

Pipeline Name\* Service\_1

Label Template\* \${COUNT} (?)

Automatic pipeline locking (?)

Automatic pipeline scheduling (?)  
Since this pipeline is based on a template, automatic/manual behaviour is determined by the template.

### Timer Settings

Cron Timer Specification (?)

Run only on new material (?)

Clicking on the icon indicated by arrow will display the following:

Services-Template

compile >> compile-job

Resources None 1

Job Timeout Use default

Run on all agents No

Tasks Artifacts Environment Variables Custom Tabs

go/service\_1\$ ant -f "build.xml" compile 2

On Cancel go/service\_1\$ ant -f "build.xml" clean 3

Run if Passed 4

conditionally, please ensure at least one of its materials has polling enabled.

The pop-up shows the extract of the template "Services-Template" configured for the

pipeline "Service\_1".

1. Shows the details of the job "compile-job" configured for the stage "compile".
2. Indicates that the working directory set for the task is "go/service\_1", which is followed by the "\$" symbol and then the command.
3. If any "On Cancel Task" has been configured, it will be indicated like this.
4. Shows the "Run If Condition" for this task.

## See also...

- [Templates - Configuration Reference](#)

# Choose when a stage runs

---

Often there are steps in your [pipeline](#) that you do not want to happen automatically. For example, you might want to keep binaries from being created for every pipeline (to prevent [running out of disk space](#)) or want to choose when your code is [deployed to production](#). Stages in Go can be marked as 'manual' just for this purpose.

You can create a manual pipeline by setting the first stage to manual.

## Example usage

---

Usage: We need a manual 'dist' stage that will create the binaries used by later stages.

- [Add a new stage](#) named 'dist' after a build stage
- Set the Stage type to manual

Stage Settings

Jobs

Environment Variables

Permissions

## Stage Settings

Stage Name\*

Dist

Stage Type: [?](#)

- On Success  
 Manual

Fetch Materials

Never Cleanup Artifacts

Clean Working Directory

\* indicates a required field

- Now, when the build stage 'build' is completed, you can manually cause Go to create the binary from the [Pipeline activity](#) page

# Run a pipeline on a schedule

---

To run a pipeline at a given time, use a timer. Timers understand a cron-like specification for when to run a pipeline.

Note that a pipeline will still schedule normally if changes are checked in. If the pipeline should only run according to the timer's schedule then you should also set a manual approval for the first stage of the pipeline to stop it from automatically scheduling when materials change. This can be also be achieved by un-checking the option "Automatic pipeline scheduling" shown in the screenshot below.

The timer is similar to a manually triggered pipeline in many ways. But it does not fetch the latest revision of the materials when it runs. It uses the last available revision that it knows of.

## Configure through the UI

---

To configure the timer in the UI, navigate to the General Options section of the pipeline. For example, a timer that is configured as shown in the screenshot would run the pipeline at 10pm on weekdays. An option called "Run only on new material" is also available in this form. Selecting this option ensures that the pipeline will get triggered on the specified schedule only if materials have changed since the last run of this pipeline. For example, if there are no new commits since the last run, future runs will be skipped until new commits or until the pipeline is forced to run (using trigger-with-options) with an older commit. This option is typically useful when "Automatic pipeline scheduling" is turned off.

[General Options](#)[Project Management](#)[Materials](#)[Stages](#)[Environment Variables](#)[Parameters](#)

## Basic Settings

Pipeline Name\*

Deploy-Prod

Label Template\*

\${COUNT}

 Automatic pipeline locking Automatic pipeline scheduling

## Timer Settings

Cron Timer Specification

0 0 22 ? \* MON-FRI

 Run only on new material*For this pipeline to schedule conditionally, please ensure at least one of its materials has polling enabled.*

\* indicates a required field

[RESET](#)[SAVE](#)

# Configure through the XML

The following xml config corresponds to the UI example above.

```
<pipeline name="nightly">
<timer onlyOnChanges="true">0 0 22 ? * MON-FRI</timer>
<materials>
    ...
</materials>
<stages>
    <stage name="compile">
        <approval type="manual"/>
    ...
</pipeline>
```

For more information see [`<timer>`](#)

# Job Timeout

---

## Introduction

---

Go can be configured to automatically cancel jobs that do not generate any console output for a period of time. Default Job timeout options include:

- Never : Jobs will never be timed out by default. You can override this behavior when configuring the job in the job editor
- Timeout after a period of inactivity : A job will be cancelled if it did not have any console output for a period of time (in minutes)

When a job is timed out, the onCancel task for the job will be triggered.

## Configuration

---

### Specify default job timeout at the server level

You must be logged in as an admin user to configure this step.

1. Navigate to the Admin section on the Go dashboard.
2. Navigate to Server configuration
3. Navigate to the pipeline management sub-section

#### Pipeline Management

The screenshot shows the 'Pipeline Management' configuration page. At the top, there is a field for 'Artifacts Directory Location' containing the value 'artifacts'. Below this, there is a section for 'Auto delete old artifacts:' with two radio button options: 'Never' (unchecked) and 'When available disk space is less than [10] GB, and keep deleting until [25] GB is available' (checked). In the middle of the page, there is a section for 'Default Job Timeout' with two radio button options: 'Never timeout' (unchecked) and 'Cancel after [30] minute(s) of inactivity' (checked). A red arrow points to the '30' input field. At the bottom of the page, there are two buttons: 'SAVE' (blue) and 'RESET' (grey).

4. Enter the default timeout for a job.

## Configure timeout behavior for a job

You must be logged in as an admin user to configure this step.

You can configure timeouts for each job if the timeout behavior needs to be different from the default timeout.

1. Navigate to the Admin section on the Go dashboard.
2. Navigate to the job settings page for the job.

The screenshot shows the Go Admin interface with the following details:

- Project Path:** ant-project » defaultStage » defaultJob
- Job Settings Tab:** The active tab is "Job Settings". Other tabs include Tasks, Artifacts, Environment Variables, and Custom Tabs.
- Job Name:** defaultJob
- Resources:** An input field for comma-separated lists, currently empty.
- Job Timeout:** A section with three options:
  - Never
  - Use default (60 minute(s))
  - Cancel after  minute(s) of inactivityA red arrow points to the "Use default (60 minute(s))" option.
- Run on all agents:** A checkbox that is unchecked.
- Required Field:** A note at the bottom left indicates that an asterisk (\*) indicates a required field.
- Buttons:** At the bottom are "SAVE" and "RESET" buttons.

3. Choose the desired timeout behavior. You can choose to never timeout the job, provide a specific value or use the default job timeout.

## Also see...

- [Adding a job](#)
- [Clean up after cancelling a task](#)

# Managing Users

Go's user management features allow you to control access to Go and grant role-based permissions.

All user management features depend on an [authentication mechanism](#) having been configured in Go. Please ensure you have at least one [authentication mechanism](#) enabled before attempting to use any of the features mentioned in this chapter.

## Adding Users

1. Navigate to the Admin section
2. Click on the "User Summary" tab
3. Click the "Add User" button

The screenshot shows the Go Admin interface with the following details:

- Top navigation bar: Pipelines, Templates, Config XML, Server Configuration, User Summary (selected), User Roles, OAuth Clients, OAuth Enabled Gadget Providers, Backup.
- User Summary tab: Enabled: 42, Disabled: 33, License Usage: 42/99999.
- Action buttons: ENABLE, DISABLE, ROLES ▾, ADD USER (highlighted with a red arrow).
- Search/filter row: Username ▾, Roles ▾, Aliases ▾, Admin ▾, Email ▾, Enabled ▾.

1. Enter a name/email to search for (minimum 2 characters) and click "Search"
2. This will perform a search across all authentication mechanisms configured (password file and/or LDAP)
3. From the list of results, select the user to add and click "Add User"

The screenshot shows a modal window titled "Add User". At the top, there is a search bar with the placeholder "Search for User:" and a search button labeled "SEARCH". Below the search bar is a table with the following columns: "Select", "User Name", "Full Name", "Email Address", and "Source". The table contains eight rows, each representing a user entry from a LDAP source. The users listed are sajay, gsmith, dsmith, bsmith, jensmith, ssmith, msmith, and jhighsmi. The "Email Address" column shows entries like "\*\*\*\*\*@thoughtworks.com". The "Source" column for all users is "LDAP". The last row of the table has a blue circular icon next to the "Select" column. At the bottom right of the modal are two buttons: "ADD USER" and "CLOSE".

Select	User Name	Full Name	Email Address	Source
<input type="radio"/>	sajay	Smitha Ajay	*****@thoughtworks.com	LDAP
<input type="radio"/>	gsmith	Gregory N. Smith	*****@thoughtworks.com	LDAP
<input type="radio"/>	dsmith	Darren Michael Smith	*****@thoughtworks.com	LDAP
<input type="radio"/>	bsmith	Brian Smith	*****@thoughtworks.com	LDAP
<input type="radio"/>	jensmith	Jennifer Smith	*****@thoughtworks.com	LDAP
<input type="radio"/>	ssmith	Stefan Smith	*****@thoughtworks.com	LDAP
<input type="radio"/>	msmith	Mike J Smith	*****@thoughtworks.com	LDAP
<input checked="" type="radio"/>	jhighsmi	Jim Highsmith	*****@thoughtworks.com	LDAP

## Assigning Roles

Roles allow you to group a set of users with similar functional duties and grant them a common set of permissions.

For example, you may have 3 pipelines configured as part of your workflow -- build, acceptance and deploy. Your team may consist of 6 developers and 2 testers. With roles, you can group all 6 of your developers into a role called "developers" and your 2 testers into a role called "testers". You'd then assign the following permissions to your pipelines:

- build: Auto triggered pipeline with approval permissions granted to both developers and testers
- acceptance: Auto triggered pipeline with approval permissions granted to testers only
- deploy: Manually triggered pipeline with approval permissions granted to testers only

With this setup, your entire team has visibility into what each other is doing, but you have controls around which role can do what.

### To assign roles to users:

1. Navigate to the Admin section
2. Click on the "User Summary" tab

3. Select the users you want assign roles to, or remove roles from
4. Click the "Roles" button to see a list of roles
5. Check/un-check the roles you want to assign/remove from the selected users and click "Apply"
6. Alternately, you can create a new role to apply to the selected users by typing in the name of a role in the input box and clicking "Add"

Username	Role	Admin
akrishna	admins	Yes
anushr	admins	Yes
<b>lwan</b>	admins	Yes
bingzhao	admins	Yes
both	*	No
cjbriese		No

## Managing 'Go System Administrator' privilege

'Go System Administrator' has access to all administrative functions, and has operational access to all parts of a Go installation.

User management page allows you to assign admin privileges to or revoke admin privileges from selected users.

This control allows you to modify admin privileges for users(not for roles). Checking 'Go System Administrator' for selected users adds them directly to \ configuration tag. This control is disabled when one or more of the selected users have implicit admin privilege through role(s).

### Assign/Revoke 'Go System Administrator' privilege:

1. Navigate to the Admin section
2. Click on the "User Summary" tab

3. Select the users you want assign/revoke 'Go System Administrator' privilege.
4. Click the "Roles" button to load 'Go System Administrator' control
5. Check/un-check the 'Go System Administrator' checkbox and click "Apply"

# Authentication

---

Go was built from the bottom up with security in mind. Go server provides both an http service and an https service by default. The http service listens on port 8153 and the https service listens on port 8154.

By default, Go does not require users to authenticate. However we provide two mechanisms for you to force users to authenticate if you wish to. You can create a password file (in standard Apache htaccess syntax) to authenticate log in requests. Alternatively Go can authenticate against LDAP or ActiveDirectory servers.

You can use both password file and LDAP/ActiveDirectory authentication at the same time. In this case Go will first try and authenticate you against the password file. If it cannot find your username, or if it finds that your username and password do not match, it will try LDAP/AD next. This can be very useful if you need a read-only user that can be used by scripts, and you do not want to add this user to your LDAP.

## File Based Authentication

---

The simplest way to authenticate people is to create a password file for Go to use. This is just a plain text file with the following format:

```
[username]:[password hashed with SHA1 and encoded with base 64]
```

If your SHA1 algorithm and base 64 encoding works properly, the password "badger" should come out as "ThmbShxAtJepX80c2JY1FzOEmUk=".

You can put as many username/hashed password pairs as you like -- use a new line for each one.

To configure Go to use a password file for authentication:

1. Login to Go as an admin
2. Navigate to the "Admin" section
3. Click on the "Server Configuration" tab
4. Fill out the "Password File Settings" field under the "User Management" section

## User Management

Allow users that exist in LDAP or in the password file to log into Go, even if they haven't been explicitly added to Go.

### LDAP Settings

URI\*

Manager DN\*

Manager Password  
 CHECK LDAP

Search Base\*

Search Filter\*

### Password File Settings

Password File Path  
 

Go should pick up this change immediately and start authenticating new users (note that anybody already using Go will be required to re-authenticate).

The file format for the password file is the [standard one for Java Properties](#), which means that spaces, the equals sign, and the colon are special characters in the username and must be escaped with a backslash.

## Generating passwords using htpasswd

You can use the [htpasswd program from Apache](#) to manage your password file. **You must use the -s option with htpasswd to force it to use SHA1 encoding for the password.** So for example, you can use the following command to create a password file called "passwd" and put the password for the user "user" in it:

```
htpasswd -c -s passwd user
```

## htpasswd on Windows

htpasswd is not available on windows, but there are plenty of [websites](#) that perform the encryption for free. Make sure you use the SHA1 algorithm.

## **htpasswd on Mac OSX**

htpasswd is already installed by default on Mac OSX.

## **htpasswd on Linux**

Debian based distributions (e.g. Ubuntu) htpasswd can be installed from the apache2-utils

```
$ apt-get install apache2-utils
```

## **Generating passwords using python**

Another option is to use the following command (assumes python is installed on your system)

```
$ python -c "import sha;from base64 import b64encode;print b64encode(sha.new('my-pass').digest())"
```

## **LDAP/ActiveDirectory Authentication**

---

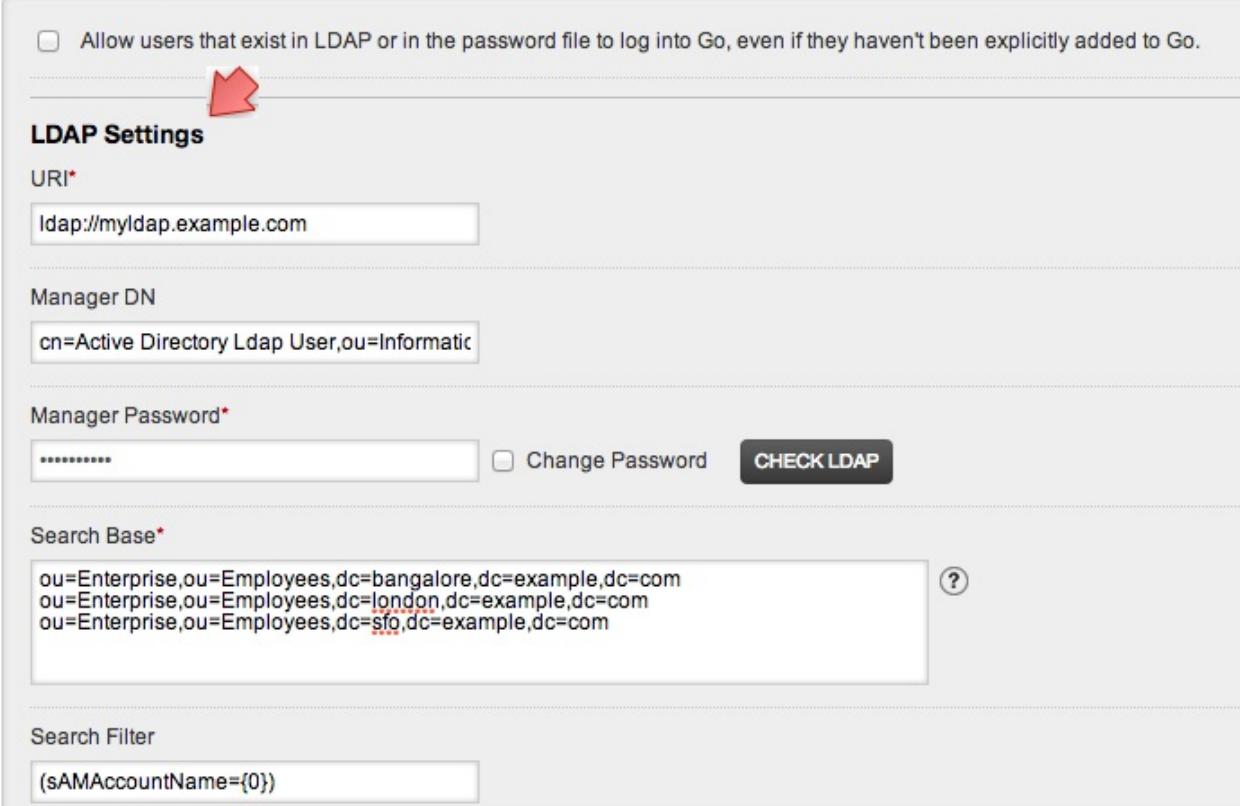
Go can authenticate against an LDAP or Active Directory (AD) server. Go uses the standard JNDI APIs to access LDAP/AD, using the well known Acegi Security framework. Go uses "bind" authentication to authenticate directly to the LDAP/AD server.

Note that LDAP/AD Authentication can be complex to configure. We highly recommend that you work with your network administration staff to configure this feature.

To configure Go to use LDAP/AD for authentication:

1. Login to Go as an admin
2. Navigate to the "Admin" section
3. Click on the "Server Configuration" tab
4. Fill out the "LDAP Settings" under the "User Management" section

## User Management



Allow users that exist in LDAP or in the password file to log into Go, even if they haven't been explicitly added to Go.

### LDAP Settings

URI\*  
ldap://myldap.example.com

Manager DN  
cn=Active Directory Ldap User,ou=Informatic

Manager Password\*  
\*\*\*\*\*  Change Password **CHECK LDAP**

Search Base\*  
ou=Enterprise,ou=Employees,dc=bangalore,dc=example,dc=com  
ou=Enterprise,ou=Employees,dc=london,dc=example,dc=com  
ou=Enterprise,ou=Employees,dc=sfo,dc=example,dc=com (?)

Search Filter  
(sAMAccountName={0})

The **Manager DN** is the LDAP/AD manager user's DN, used to connect to the LDAP/AD server.

The **Manager Password** is the LDAP/AD manager password, used to connect to the LDAP/AD server. Use the 'change password' checkbox to edit the password.

The **Search Base** is the name of the context or object to search in for the user record. If you have more than one search base, please separate each of them with a new line.

The **Search Filter** is the expression used in the user search. It is an LDAP search filter as defined in [RFC 2254](#) with optional parameters -- in this case, the username is the only parameter. An example might be:

(uid={0})

which would search for a username match on the uid attribute, or

(sAMAccountName={0})

which would search for a username match on the sAMAccountName attribute (for

ActiveDirectory users)

Click on Check LDAP button to check if your LDAP configuration is correct. This will bind to the LDAP server using the credentials provided in Manager DN and Manager Password.

Check LDAP will report an error if Search Base contains invalid **dc** information. However, it will not detect invalid **ou**

The authentication operation has two steps: firstly, Go uses the Manager DN and Manager Password supplied to search for the user using the searchBase and searchFilter attributes. Go will search subtrees and time out after five seconds. Go then uses the DN returned to attempt to bind to LDAP/AD using the username and password supplied by the user.

Note that Go doesn't retrieve any further information from LDAP/AD such as roles, groups or email address. It simply gets the user's CN.

If multiple search bases are configured, Go server will look for the specified user in each search base, one after the other. It will stop searching when it finds the information in a search base. In case any of the search bases are invalid, Go server will log this information in the server log, but continue searching in the remaining search bases.

## Controlling User Access

---

Once a user is authenticated, Go checks to see if he is an existing user or a new user (logging in for the first time). If a new user, there are two behaviors Go can operate under:

- Automatically register the new user in Go and continue with the login process. This option has implications on licensing because auto-registering any new user who is in LDAP might cause you to run over your license limit. So keep that in mind when using this option.
- Deny access to the user if not already a registered Go user. New users will have to be explicitly added by an admin.

To switch the mode in which the Go Server operates:

1. Login to Go as an admin
2. Navigate to the "Admin" section
3. Click on the "Server Configuration" tab

- Set the "Allow users that exist in LDAP or in the password file to log into Go, even if they haven't been explicitly added to Go" checkbox

## User Management



The screenshot shows the 'User Management' configuration page. At the top, there is a checkbox labeled 'Allow users that exist in LDAP or in the password file to log into Go, even if they haven't been explicitly added to Go.' A red arrow points to this checkbox. Below it, there is a section titled 'LDAP Settings' containing fields for 'URI\*', 'Manager DN\*', 'Manager Password', 'Search Base\*', and 'Search Filter\*'. There is also a 'CHECK LDAP' button next to the Manager Password field. Below this, there is a section titled 'Password File Settings' with a 'Password File Path' field.

## Common errors

Below are few of the common errors you might encounter while integrating with an authentication provider

### Bad credentials

- Invalid username/password combination. Please check if the combination is valid and try again.

### User [username] not found in directory

- A user with [username] is not found in LDAP. Please check with your LDAP administrator to verify if the user exists.
- Check with your Go Administrator to verify that the user with [username] exists in the LDAP search base configured in Go.

## **Empty username not allowed**

- The user has supplied an empty username. Please enter a valid username in the field.

## **Failed to authenticate with your authentication provider. Please check if your authentication provider is up and available to serve requests.**

- Your LDAP server could not be reached by Go Server. Please check with your LDAP Administrator to resolve connectivity issues, if one exists, between Go Server and LDAP.
- Please check with your Go Administrator to verify that the LDAP configuration is correct. Also check the Go Server logs for errors.

## **User license limit exceeded, please contact the administrator**

- This error is displayed when the number of users logged into Go has exceeded the number permitted by the license. This typically happens when an existing license expires. It can also happen if a license which allowed certain number of users has been replaced by another which allows a lesser number of users.

## **Your account has been disabled by the administrator**

- This error is displayed when the user trying to log into Go has been disabled by the administrator. Please check with your Go Administrator.

# Authorization

With no security switched on, there is of course no authorization either. Once you have configured security, the default is that any user can perform any operation. However you can get Go to limit certain operations to particular Users or Roles, and manage membership of those Roles.

## Administrators

Go allows you to restrict the users who can perform certain functions. Administrators is a special role that allows its members to perform any action in Go. Specifying administrators is optional -- without it, all users are automatically made administrators.

If administrators are specified, only they can perform the following actions:

- Access the "administration" tab
- Add/Edit Pipeline Templates
- Enable agents
- Add / remove agent resources

Users can be made administrators from the "User Summary" tab in the "Admin" section.

The screenshot shows the 'User Summary' tab in the 'Administration' section of the Go interface. The top navigation bar includes tabs for Pipelines, Templates, Config XML, Server Configuration, User Summary (which is active), User Roles, OAuth Clients, and OAuth Enabled Gadget Providers. Below the tabs, there are buttons for ENABLE, DISABLE, and ROLES. The ROLES dropdown is open, showing the 'Go System Administrator' role selected. The main table lists users with columns for Username, Email, and Enabled status. The user 'aantony' is highlighted with a red arrow pointing to the 'ENABLE' button, which has a checked checkbox. A red arrow also points to the 'ADD' button at the bottom right of the roles list.

To give admin privileges to users and/or roles via "Config xml", please refer to the example in the section below, where members of the "go\_admin" role (jhumble and qiao), along with the user chris, can administer Go.

# Role-based security

You can define roles that can be used anywhere that authorization is required. A role is just a group of users. Administrators can add users to a new or existing role from the "User Summary" tab in the "Admin" section. Here, you can select any number of users and assign a new or existing role to them. In this example, user "aantony" is being added to the role "analyst"

The screenshot shows the Go Admin interface with the 'User Summary' tab selected. A modal dialog is open for the user 'aantony'. The modal lists roles: 'Go System Administrator' (highlighted in yellow), 'admin', 'analyst' (with a checked checkbox), and 'developer'. At the bottom of the modal is a blue 'APPLY' button. Red arrows point to the 'analyst' checkbox and the 'APPLY' button.

For power users, here's how you would configure roles via "Config XML":

```
<cruise>
  <server>
    <license ... />
    <security>
      <passwordFile path="/etc/go-server/passwords.properties" />

      <roles>
        <role name="qa">
          <user>dyang</user>
          <user>pavan</user>
        </role>
        <role name="go_admin">
          <user>jhumble</user>
          <user>qiao</user>
        </role>
      </roles>

      <admins>
        <role>go_admin</role>
        <user>chris</user>
      </admins>
    </security>
  </server>
</cruise>
```

In this example, the "qa" role has two users: dyang and pavan. The "go\_admin" role also has two users: jhumble and qiao.

## Specifying permissions for pipeline groups

---

Go allows you to group pipelines together. If you define pipeline groups, you can specify who is able to view or operate those groups. To do this, you configure permissions to the pipeline group. System administrators will continue to have full access to the pipeline group even if they have not been explicitly granted permissions.

The "**view**" permission allows users to view the pipeline. It does not give permission to trigger pipelines, approve stages, or re-run stages. In the below example, the users "akrishna" and "aantony" can view the pipelines in this group, but they cannot perform any operations on it.

The "**operate**" permission allows users to trigger pipelines and its stages. In the below example, the role "developer" is being granted the operate permission and will be able to trigger pipelines and its stages within this group.

The "**admin**" permission makes the user a [Pipeline Group Administrator](#) allowing him to view, operate and administer the pipeline group. In the below example, role "admins" has been granted this permission.

Note that it is possible to give a user or role only the operate permission. In the example below, the user "bot" only has operate permission. That means they can not view the pipeline, they can only operate it. This can be used to enable a script to operate on pipelines via the APIs without letting that user access any other features of Go.

To edit the permissions for a pipeline group, navigate to the "Pipelines" tab on the "Admin" section:

## Deploy-Train-Apps

[Edit](#) [Delete](#)

Pipeline	Actions				
app-1	<a href="#">Edit</a>	<a href="#">Move to ▾</a>	<a href="#">Clone</a>	<a href="#">Delete</a>	<a href="#">Extract Template</a>
app-2	<a href="#">Edit</a>	<a href="#">Move to ▾</a>	<a href="#">Clone</a>	<a href="#">Delete</a>	<a href="#">Extract Template</a>
app-3	<a href="#">Edit</a>	<a href="#">Move to ▾</a>	<a href="#">Clone</a>	<a href="#">Delete</a>	<a href="#">Extract Template</a>

[+ Create a new pipeline within this group](#)

Then, click the "Edit" link for the pipeline group you want to manage permissions for:

### User Permissions ?

All system administrators and pipeline group administrators belonging to roles configured below will automatically have admin rights (this cannot be overridden).

Name	View	Operate	Admin	Remove
aantony	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<a href="#">×</a>
akrishna	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<a href="#">×</a>
bot	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<a href="#">×</a>

[+ Add user permission](#)

### Role Permissions ?

Name	View	Operate	Admin	Remove
admins	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<a href="#">×</a>
dev	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<a href="#">×</a>
developer	<a href="#">+</a>			

[SAVE](#)

[RESET](#)

If no authorization is defined for a pipeline group, all Go users will have view and operate permissions to that group.

For power users, here's how you would configure permissions via "Config XML":

```
<pipelines group="Shine">
  <authorization>
    <view>
      <user>aantony</user>
      <user>akrishna</user>
      <role>developer</role>
    </view>
```

```
<operate>
  <user>bot</user>
  <role>developer</role>
</operate>
<admins>
  <role>admins</role>
</admins>
</authorization>
...
</pipelines>
```

## Adding authorization to approvals

In Go, it is possible to specify [manual approvals](#) between stages. You can also specify which user is allowed to trigger manual approvals.

The authorization can be inherited from the pipeline group this pipeline belongs to. But defining specific permissions overrides this. In the example below, only members of the role "admin", and the user "goleys", can trigger the approval.

### » test\_pipeline » defaultStage

Stage Settings    Jobs    Environment Variables    Permissions

#### Permissions

All system administrators and pipeline group administrators can operate on this stage (this cannot be overridden).

For this stage:  Inherit from the pipeline group  Specify locally

*(A large red arrow points to the 'Specify locally' radio button.)*

**Info:** The pipeline group that this pipeline belongs to has permissions configured. You can add only those users and roles that have permissions to operate on this pipeline group.

**Users**

goleys X  
 X

[+ Add](#)

**Roles**

admin X  
 X

[+ Add](#)

For power users, here's how you would configure authorization for approvals for a stage via "Config XML":

```
<stage name="defaultStage">

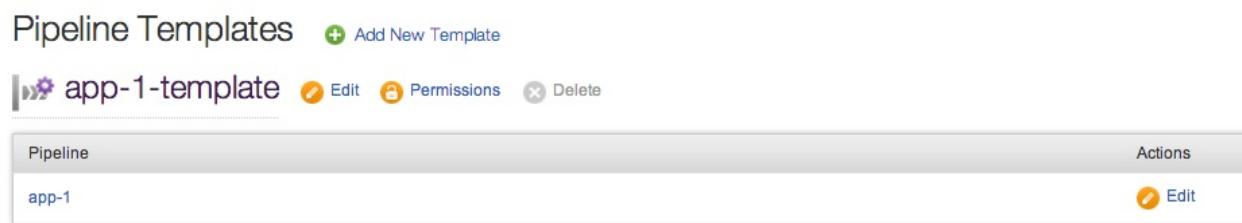
    <approval type="manual">
        <authorization>
            <role>admin</role>
            <user>goleys</user>
        </authorization>
    </approval>

    <jobs>
        <job name="deploy">
            <resources>
                <resource>uat</resource>
            </resources>
            <tasks>
                <ant target="deploy" />
            </tasks>
        </job>
    </jobs>
</stage>
```

## Specifying permissions for templates

A Go Administrator can make any user a template administrator for a specific template. As a template administrator, a user can now view and edit the template to which he has permissions.

To edit the permissions for a template, navigate to the "Templates" tab on the "Admin" section:



The screenshot shows the "Pipeline Templates" section of the Go Admin interface. A template named "app-1-template" is selected. At the top, there are buttons for "Add New Template" and "Edit", "Permissions", and "Delete" links. Below the header, there is a table with one row labeled "app-1". On the far right of this row is an "Edit" link. The entire row is highlighted with a light gray background.

Then, click the "Permissions" link for the template you want to manage permissions for:



The screenshot shows the "User Permissions" dialog box. It has a table with a single row. The first column is "Name" and the second column is "Remove". There is a delete icon (an 'X') at the end of the "Remove" column. At the bottom left is a "Add user permission" button.

For power users, here's how you would configure permissions via "Config XML":

```
<templates>
  <pipeline name="app-1-template">
    <authorization>
      <admins>
        <user>operate</user>
      </admins>
    </authorization>
    ...
  </pipeline>
</templates>
```

## Also See

- [Delegating group administration](#)

# Delegating Group Administration

A Go Administrator can authorize users and roles to be administrators for Pipeline Groups. These group administrators have certain privileges which are explained in the section "Privileges of a Group Administrator".

## Steps to assign Group Administrators

To assign a user as a group administrator:

1. Click on "Pipelines" tab on the **Admin** section
2. Locate the group you want to assign a group administrator to
3. Click the "Edit" link for that group
4. Here, you can define permissions for users and roles

### User Permissions ?

All system administrators and pipeline group administrators belonging to roles configured below will automatically have admin rights (this cannot be overridden).

Name	View	Operate	Admin	Remove
jez	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="button" value="x"/>

[+ Add user permission](#)

Name	View	Operate	Admin	Remove
groupAdminRole	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="button" value="x"/>

[+ Add role permission](#)

**SAVE**   **RESET**

In the above screenshot, the Go admin has delegated group admin privileges to a user "jez" and all users defined under the role "groupAdminRole". The privileges of a Group Administrator have been described in the next section.

For power users, here's how you'd assign the same permissions via Config XML:

```
<pipelines group="studios">
    <authorization>
        <admins>
            <user> jez </user>
```

```
<role> groupAdminRole </role>
</admins>
</authorization>
<pipeline name="go_pipeline">
...
</pipeline>
</pipelines>
```

## Privileges of a Group Administrator

---

As a group administrator of a pipeline group, a user is privileged to:

- View and operate (trigger, rerun stages etc.) all the pipelines in this group.
- Add other group admins to this group
- Authorize users/roles with 'view' and 'operate' permissions for this pipeline group.
- Add and Delete pipelines to/from the group.
- Add a pipeline using the "Add New Pipeline" wizard, but only to the groups he is allowed to administer.
- Edit pipelines belonging to the group. Which includes renaming, adding, deleting and modifying stages and jobs.
- Restfully view and operate (trigger, rerun stages etc.) all the pipelines in this group.
- Restfully edit the pipelines belonging to this group.

**Note:** A group administrator can access "Pipelines" and "Config XML" tabs on the Administration page to [view and edit his/her pipeline groups](#). He/She cannot access Server Configuration or perform user management. While a group administrator cannot access Pipeline Templates either, they can use existing templates for pipelines within their pipeline group.

# Pipeline Groups Administration

---

Pipeline Group Administrators in Go can add, remove and edit pipelines in their respective pipeline groups. They can do these operations via the UI as well by editing the config XML of the pipeline group.

## Administration using UI

---

The administration page for a pipeline group administrator looks as follows. The controls on the "Pipelines" tab allows her to edit, clone, delete and move any pipeline.

The screenshot shows the administration interface for pipeline groups. At the top, there's a navigation bar with tabs for "Administration", "Pipelines", and "Config XML". The "Pipelines" tab is selected. Below the navigation bar, there are two sections, each representing a pipeline group:

- go-project**: This section shows a single pipeline named "build-go". It includes an "Edit" button and actions for "Move to", "Clone", and "Delete". A link to "Create a new pipeline within this group" is also present.
- mingle-project**: This section shows a single pipeline named "build-mingle". It includes an "Edit" button and actions for "Move to", "Clone", and "Delete". A link to "Create a new pipeline within this group" is also present.

## Administration using XML

---

The tab "Config XML" shows the XML snippets of each pipeline group. The user can toggle between different pipeline groups and view/edit them one at a time.

## Pipeline Groups

EDIT

**go-project**

mingle-project

```
<pipelines group="go-project">
  <authorization>
    <view>
      <user>jez</user>
    </view>
    <admins>
      <user>jez</user>
    </admins>
  </authorization>
  <pipeline name="build-go">
    <materials>
      <hg url="../manual-testing/ant_hg/dummy" dest="dest_dir" />
    </materials>
    <stage name="compile-go">
      <jobs>
        <job name="core">
          <tasks>
            <exec command="ls" />
          </tasks>
        </job>
      </jobs>
    </stage>
  </pipeline>
</pipelines>
```

# Publishing artifacts

---

When executing a job on an agent there are often artifacts created that we need to keep around. For example, JUnit creates xml reports that Go is able to parse in order to help you [understand why the build is broken](#). You can use Go with any XUnit style xml reports. Or you might create a flash video of your UI tests that we want displayed in Go. You can upload any html file from your build and view it in Go.

To publish artifacts you add a an [`<artifact>`](#) to the job configuration. More information can be found on the [Managing artifacts and reports](#) page.

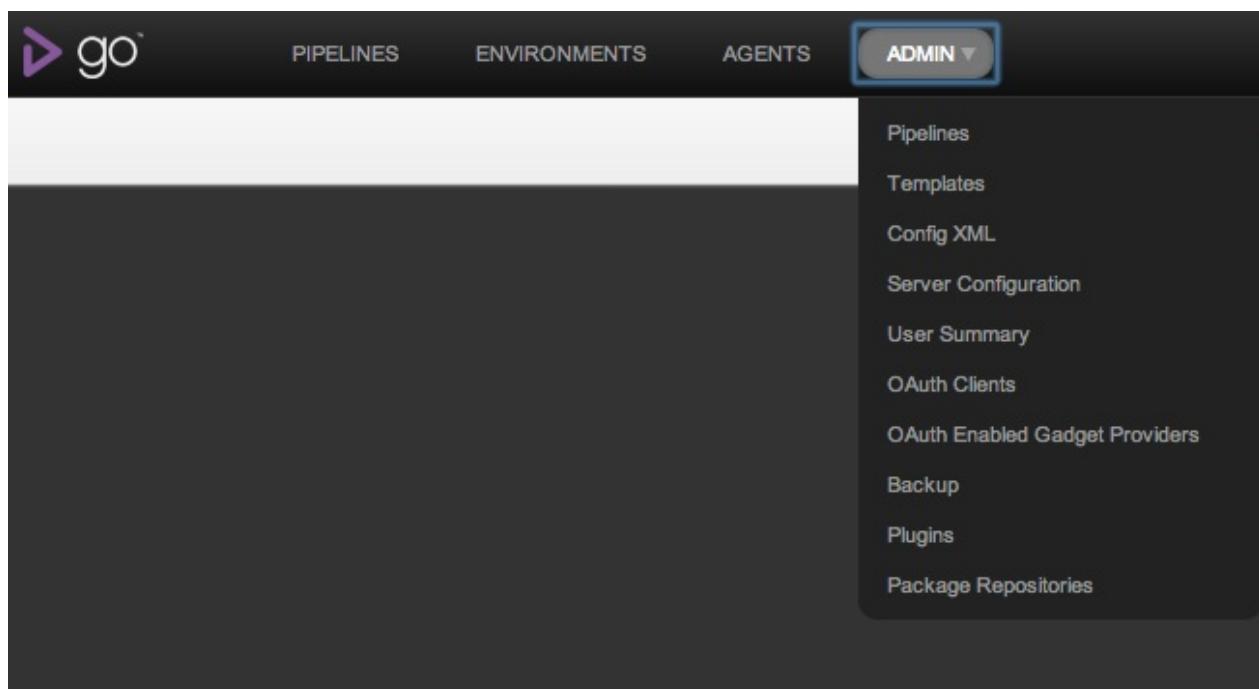
## Example usages

---

### Uploading JUnit xml reports

We are going to assume that the JUnit test reports are being placed in the "target/reports" folder.

Click on the **Administration** tab



Click on your pipeline

Pipeline Actions

- app-1: Edit, Move to, Clone, Delete, Extract Template
- app-2: Edit, Move to, Clone, Delete, Extract Template
- app-3: Edit, Move to, Clone, Delete, Extract Template

[+ Create a new pipeline within this group](#)

For each job that runs JUnit:

- Click on the job name to edit job config

app-1 » build

Job	Resources	Run on all	Remove
test		No	<a href="#">x</a>

[+ Add new job](#)

- Add the source of the test artifact. For tests, choose the type of artifact as Test artifact

app-1 » build » test

Artifacts

Source	Destination	Type
project/reports		Test Artifact
project/test_reports		Test Artifact

[+ Add](#)

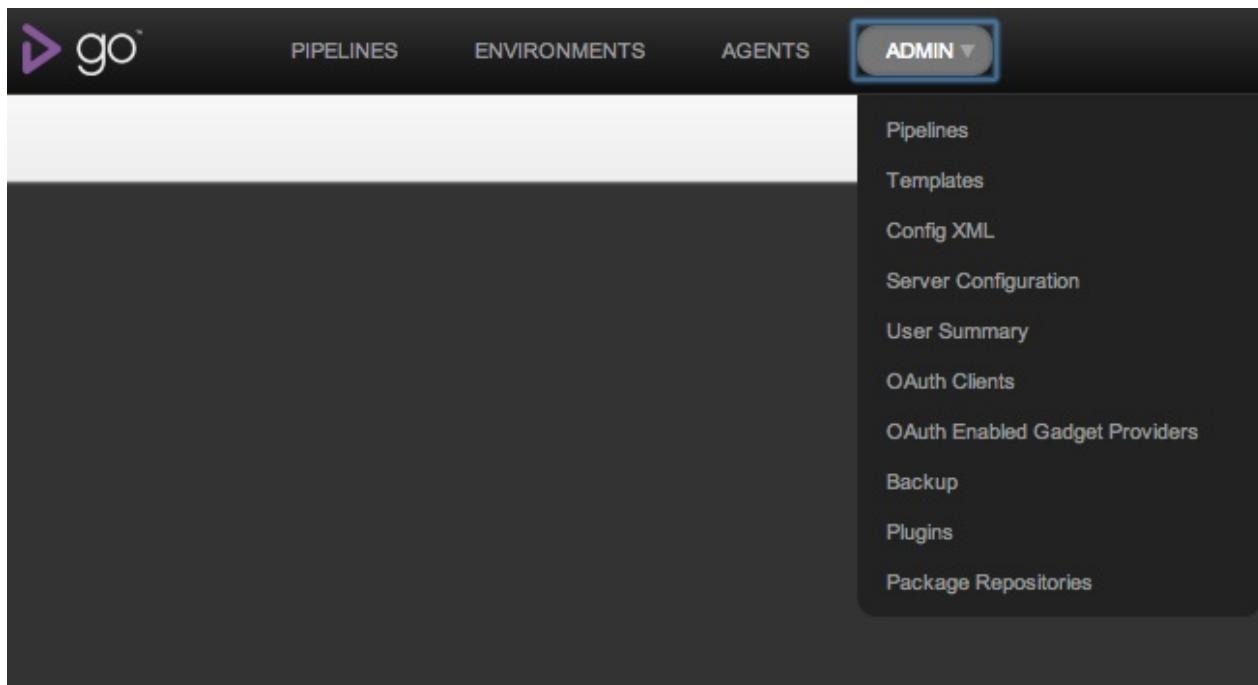
[SAVE](#) [RESET](#)

- Click "Save"

## Uploading a flash video and displaying it as a sub-tab

We are going to assume that the flash file, and the html file referencing it, are being created in the "target/reports" folder.

Click on the Administration tab



Click on your pipeline

The screenshot shows the 'Deploy-Train-Apps' pipeline group. It lists three pipelines:

- app-1
- app-2
- app-3

Each pipeline entry includes the following actions:

- Edit
- Move to
- Clone
- Delete
- Extract Template

At the bottom left, there is a green plus icon with the text "Create a new pipeline within this group".

For each job that creates a flash video

- Click on the job name to edit the job config

The screenshot shows the 'app-1 » build' stage configuration. It displays the following structure:

- app-1
  - build
    - test
  - dist

The 'test' job under the 'build' stage is highlighted with a red arrow. In the 'Jobs' section, the 'test' job is listed with the following details:

Job	Resources	Run on all	Remove
test		No	X

Below the table is a green plus icon with the text "+ Add new job".

- Navigate to the "Artifacts". Add the source of the artifact. Choose the type of artifact as Build artifact. This will copy all files from the "target/reports" folder on the agent to

the "Recording" folder on Go server

The screenshot shows the Jenkins interface for configuring artifacts. At the top, there are tabs: Job Settings, Tasks, Artifacts (which is selected), Environment Variables, and Custom Tabs. Below the tabs, under the heading 'Artifacts ?', there is a form with three fields: 'Source ?' containing 'twist/reports', 'Destination ?' containing 'recording', and 'Type ?' set to 'Build Artifact'. A green '+' button labeled 'Add' is available to add more entries. At the bottom of the form are 'SAVE' and 'RESET' buttons.

- Navigate to "Custom Tabs". Add the tab name and the source of the html file. This will create a tab called "Recording" that shows the html page found at "recording/twist-recording.html" on Go server.

The screenshot shows the Jenkins interface for configuring custom tabs. At the top, there are tabs: Job Settings, Tasks, Artifacts, Environment Variables, and Custom Tabs (which is selected). Below the tabs, under the heading 'Custom Tabs ?', there is a form with two fields: 'Tab Name ?' containing 'Recording' and 'Path ?' containing 'recording/twist-recording.html'. A green '+' button labeled 'Add' is available to add more entries. At the bottom of the form are 'SAVE' and 'RESET' buttons.

- Click "Save"
- Watch the flash video as a sub-tab on the Job Details page

# Managing artifacts and reports

Because all your work is done on remote agents, Go provides a mechanism for files to be automatically uploaded to Go server following the completion of every job. These files can then be accessed via the Go server dashboard, or via the RESTful API.

## Publishing artifacts

The first step in using the artifact repository is to tell Go which files you want to publish. To do this just specify the path to the file or directory relative to the root of the source control checkout. You also need to specify where Go will publish the artifact. You can add as many files and directories as you like.

To configure an artifact:

- Navigate to **Admin → Pipelines**
- Edit the pipeline you want to configure artifacts for
- Expand the left tree navigator and click on your job
- Click on the **Artifacts** tab
- Enter the source (where the artifact will be found) and destination (where the artifact should be saved on the Go server)

The screenshot shows the Go server's pipeline configuration interface. At the top, there is a breadcrumb navigation: **>>> upstream\_pipeline » AutoStage1 » firstJob**. Below this is a navigation bar with tabs: **Job Settings**, **Tasks**, **Artifacts** (which is the active tab), and **Environment Variables**. The main area is titled **Artifacts**. It contains a table with three columns: **Source**, **Destination**, and **Type**. There are two rows in the table:

Source	Destination	Type
target/commonlib.dll	pkg	Build Artifact
		Build Artifact

A green **+ Add** button is located below the table. At the bottom of the screen are two buttons: **SAVE** and **RESET**.

For power users, here's how you would configure this via Config XML:

```
<artifacts>
```

```
<artifact src="target/commonlib.dll" dest="pkg" />  
</artifacts>
```

## Using tabs

Once your artifacts are safely in Go server's artifact repository, you can have the dashboard display them in tabs.

Go can display images, text files, or anything else that a browser will normally render in an IFrame. If you display an html page which references other resources (such as images, Flash files or whatever), so long as the resources are referenced with relative paths, they will display correctly.

This mechanism is a simple way to include reports (for example code coverage) in Go.

## Example

The console tab shows output information from all the phases of the job. This also includes information from the version control system and details regarding the artifacts created and published during the job.

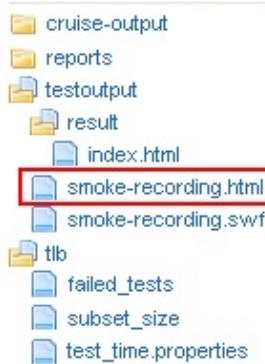


```
[cruise] Start to prepare app-Perf/52/deploy/1/P1.S1.Job1-1ce1465d-d8ba-4433-b4fd-ac5426b1ffce on  
blrstdcrspbs01.thoughtworks.com [/var/lib/cruise-agent] at Mon Jun 28 15:14:12 IST 2010  
  
[cruise] Start updating files at revision 48 from http://subversion.assembla.com/svn/tingtong/  
At revision 48.  
  
[cruise] Start to build app-Perf/52/deploy/1/P1.S1.Job1-1ce1465d-d8ba-4433-b4fd-ac5426b1ffce on  
blrstdcrspbs01.thoughtworks.com [/var/lib/cruise-agent] at Mon Jun 28 15:14:17 IST 2010  
  
[cruise] Start to execute task: <exec command="ls" args="." />. Current status: passed  
[cruise] setting environment variable 'GO_ENVIRONMENT_NAME' to value 'Demo-Perf'
```

If you produce an html page with an embedded Flash file into your artifact repository:



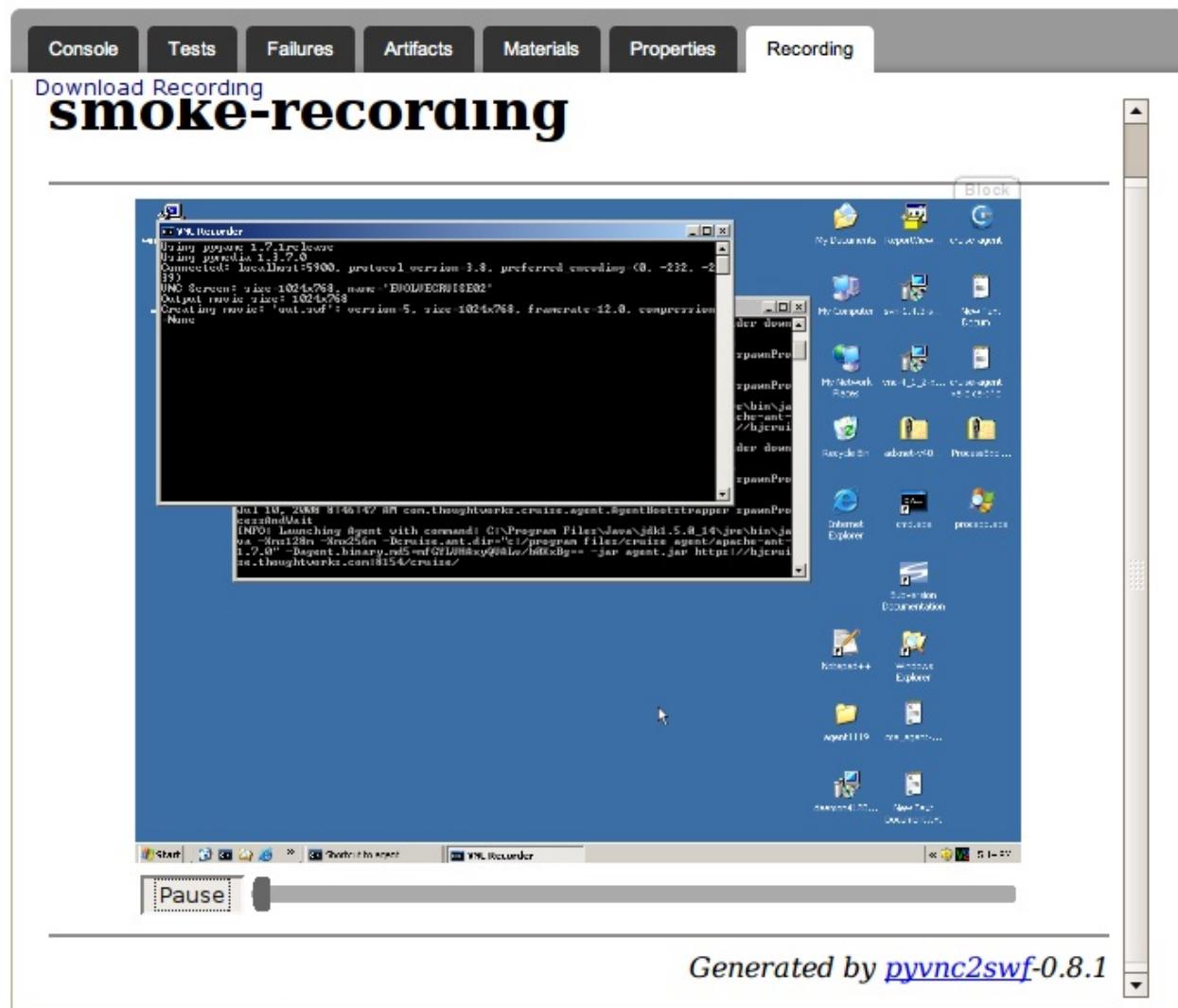
(+) Expand All (-) Collapse All



You can use the following configuration to display it in a tab:

```
<tabs>
  <tab name="Recording" path="deployment/drop/smoke/smoke-recording.html" />
</tabs>
```

Go will create a tab called "Recording" and display the contents of the file in the tab when you click on it:



# Publishing tests

Go has support for publishing tests from JUnit or NUnit test reports.

## To configure a test artifact:

- Navigate to **Admin → Pipelines**
  - Edit the pipeline you want to configure artifacts for
  - Expand the left tree navigator and click on your job
  - Click on the **Artifacts** tab
  - Enter the source (where the artifact will be found) and destination (where the artifact should be saved on the Go server)
  - From the **Type** dropdown, select **Test Artifact**

The screenshot shows the 'Artifacts' configuration screen. At the top, there are tabs for 'Job Settings', 'Tasks', 'Artifacts' (which is selected), and 'Environment Variables'. Below the tabs, there's a section for 'Source' (containing 'xstream/target/test-reports') and 'Destination' (empty). To the right of these is a 'Type' dropdown set to 'Test Artifact', with a red arrow pointing to it. There are also 'Add' and 'Remove' buttons. At the bottom are 'SAVE' and 'RESET' buttons.

For power users, here's how you would configure this via Config XML:

```
<artifacts>
  <test src="xstream/target/test-reports" />
</artifacts>
```

Go will:

- add a tab called Tests that lists the tests in the project
- add a list of the failed tests to the Failures tab
- set the properties associated with tests on the job. These include failed-test-count, ignored-test-count, test-time and total-test-count
- copy the artifacts into the repository. In this case the test reports will be copied into a new directory test-reports in the artifact repository

## RESTful API

Go publishes all of its information as resources that can be queried through http in the form of RESTful API. See the [Go integration](#) page for more information.

# Auto delete artifacts

## Introduction

Go can be configured to automatically delete artifacts if the available disk space on the server is low. Go will purge artifacts when available disk space is lower than the given value. Artifacts will be purged upto the point when available disk space is greater than a defined value.

## Configuration

### Specify artifact purge start and end limits

You must be logged in as an admin user to configure this step.

1. Navigate to the Admin section on the Go dashboard.
2. Navigate to the Pipeline Management sub-section
3. Specify when Go should begin to purge artifacts in the first edit box.
4. Specify when Go should stop purging artifacts in the second edit box.

### Pipeline Management

The screenshot shows the 'Pipeline Management' configuration page. At the top, there is a field for 'Artifacts Directory Location' containing the value 'artifacts'. Below this, under 'Auto delete old artifacts:', there is a radio button group. The 'Never' option is unselected, while the 'When available disk space is less than [ ] GB, and keep deleting until [ ] GB is available' option is selected. Two red arrows point down to the input fields for '10' and '25'. At the bottom of the page, there are 'SAVE' and 'RESET' buttons.

Artifacts Directory Location  
artifacts

Auto delete old artifacts:

Never

When available disk space is less than  GB, and keep deleting until  GB is available

Default Job Timeout

Never timeout

Cancel after  minute(s) of inactivity

**SAVE**   **RESET**

### Never delete artifacts for a stage

You must be logged in as an admin user to configure this step.

You can disallow deletion of artifacts from a particular stage so that those artifacts are excluded during deletion. This option can be set in the stage editor for a pipeline. This option can be set for stages that are important so that artifacts for the stage are preserved.

1. Navigate to the admin section on the Go dashboard.
2. Navigate to the pipelines section and choose a pipeline to edit
3. Navigate to the stage settings for the stage

The screenshot shows the Go pipeline interface for managing a project named 'my\_app'. The 'defaultStage' is selected. The 'Stage Settings' tab is active, displaying configuration options. A red arrow points to the 'Never Cleanup Artifacts' checkbox, which is checked. Other visible options include 'Stage Name' (set to 'defaultStage'), 'Stage Type' (set to 'On Success'), 'Fetch Materials' (checked), and 'Clean Working Directory' (unchecked). Buttons for 'SAVE' and 'RESET' are at the bottom.

4. Check the box 'Never Cleanup Aartifacts'

## Also see...

- [Managing artifacts and reports](#)
- [Clean up after cancelling a task](#)

# UI testing

---

Because Go installs itself as a service (Windows) or daemon (Linux) by default, getting Go agents to interact with your operating system's windowing environment can cause problems. Access to a windowing environment is usually required for testing UI applications or for driving browsers for web testing. Here's how you do it.

## Windows

---

The first step is to disable the Go agent service. To do this:

1. Log in to your Windows machine as an Administrative user.
2. Click on Start → Control Panel → Administrative Tools → Services.
3. Double click on 'Go Agent'.
4. Change the Startup Type to 'Disabled'.
5. Click 'Stop' to stop the service.
6. Click 'OK' to finish.

The next step is to start the Go agent as an application.

1. Click on Start → All Programs → Go Agent → Run Go Agent.
2. To get the Go agent to start every time you log in, copy the Run Go Agent shortcut to the Startup folder of your start menu.

## Linux

---

There are many different ways to get a Linux build agent to interact with a UI. The easiest is to use a VNC service to provide a dedicated X11 server to your agent. To do this:

1. Install the VNC server and fvwm packages for your distribution. (`aptitude install vnc4server fvwm / yum install vnc-server fvwm`)
2. Sudo to the 'go' user (`sudo su - go`) and do the rest as that user
3. Set a password for remote access to your VNC server with the command '`vncpasswd`'
4. Edit your VNC config to use fvwm and not twm as the window manager. (replace twm with fvwm in `~/.vnc/xstartup`)

5. Edit '/etc/default/go-agent' and change the line 'VNC=N' to 'VNC=Y'

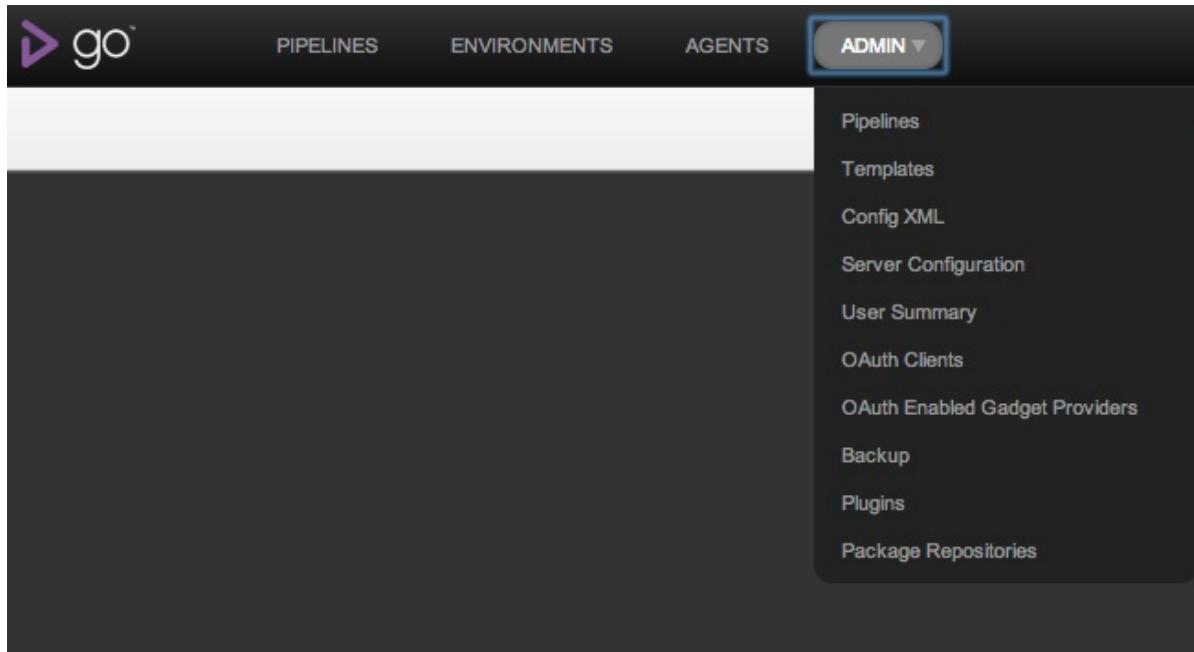
Restart your agent and it will now have access to an X11 server that you can also connect to with your favourite VNC client. The default DISPLAY that Go uses is :3

- If there are any other environmental variables that need to be set for your UI testing tools, the correct place to set these is in your /etc/default/go-agent file. Don't forget to export them!
- You can connect to your session with vncviewer to see what's going on. Use vncviewer <agent-host-name>:3
- If things appear to hang, chances are you forgot to replace twm with fvwm. twm requires you to place a window on the desktop when it starts up

# Mailhost information

In order to allow [email notifications](#), we need to tell Go information about your mailhost.

- Click on the [Administration](#) tab



- Click on the 'Server Configuration' sub-tab
- Add your mailhost information (with username and password as required)

## Email Notification

Hostname\*

Port\*

Username

Password

  Change Password

Use TLS

From\*

Admin mail\*

\* indicates a required field

- Add an Administrator email address (this account will be emailed if the Go server is [running out of disk space](#))
- Click 'Send test email' to verify the configuration is working correctly
- Click 'Save' when you're sure it's working.

# Notifications

---

It is often useful to receive an email when certain status changes occur in a stage. For example, a developer might want to know when their own check-in has broken the build. Alternatively, a manager might want an email whenever a project is deployed into production. Both of these scenarios can be covered by Notification Filters

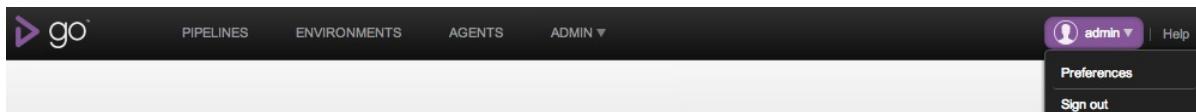
Notifications will only work if [Security](#) is enabled and [mailhost information](#) is correct.

## Example usage

---

**Usage: As a developer, I want to be notified when I break a build on "acceptance" pipeline.**

- Click on the **Preferences** tab



- Click "Edit" and enter the email address, and make sure "Enable email notification" is checked
- When I check in, my source control log in will be either "User" or "username", so enter both of those into the "My check-in aliases" box

# My Notifications

## EMAIL SETTINGS

### Email

username@servername.com

- Enable email notification

### My check-in aliases

User, username

(comma separated)

**SAVE**

**CANCEL**

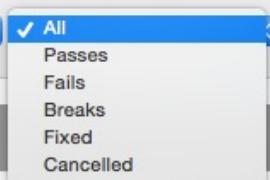
- Click "Save" to store these values
- Add a filter for the "twist-plugins" stage of "acceptance" pipeline to notify me when a check-in of mine breaks the build

## FILTERS

Pipeline	Stage	Event
acceptance	twist-plugins	<input checked="" type="checkbox"/> All Passes Fails Breaks Fixed Cancelled

Only if it contains my check-ins

**ADD**    **RESET**



# Events

You can set up notifications for different events

- All - all the runs for the stage
- Passes - the passed runs for the stage
- Fails - the stage run that failed
- Breaks - the stage run that broke the build
- Fixed - the stage run that fixed the previous failure
- Cancelled - the stage run that was cancelled

## Illustrations

- Previous build Pass, current build Fail: Event: Break

- Previous build Fail, current build Fail: Event: Fail
- Previous build Fail, current build Pass: Event: Fixed
- Previous build Pass, current build Pass: Event: Pass

Notification Filters				
Pipeline	Stage	Event	Check-ins Matcher	
acceptance	twist-plugins	Breaks	Mine	<button>DELETE</button>
plugins	build	Fails	Mine	<button>DELETE</button>
distributions-all	upload-installers	Passes	All	<button>DELETE</button>
regression	[Any Stage]	All	Mine	<button>DELETE</button>

I'll be emailed whenever the "twist-plugins" stage of "acceptance" pipeline breaks due to my check-in

I'll be emailed whenever the "build" stage of "plugins" pipeline fails due to my check-in

I'll be emailed whenever the "upload-installers" stage of "distributions-all" pipeline passes for any check-in

I'll be emailed on all events for any stage of "regression" pipeline for my check-in

Users can also select to get notifications for a particular (or all) event on any stage of any pipeline.

Notification Filters				
Pipeline	Stage	Event	Check-ins Matcher	
[Any Pipeline]	[Any Stage]	Fails	Mine	<button>DELETE</button>

I'll be emailed whenever any stage of any pipeline fails due to my check-in

# TFS Material configuration

---

You can use TFS SCM as a material for your pipeline. Go server and agent uses [TFS Java SDK v10](#) by default. The TFS SDK is packaged with Go; no additional configuration is required.

You will need to configure the following to add a TFS material:

- Material Name: An optional name for the material. This name can be used to set the TFS revision in the pipeline label.
- URL: Set the url for the TFS collection. Ex:  
<http://41.42.43.44:8080/tfs/DefaultCollection>
- Domain: Domain name for TFS authentication credentials. Should be domain for the TFS account. Ex: corporate\_thoughtworks
- Username: The user has to be a collection administrator for the TFS collection used in URL.
- Password: Provide the password for the TFS account
- Project Path: Enter the project path within the collection. You can specify paths of sub folders to create materials for each component. Ex: \$/test/component or \$/my\_application
- Check connection: You can use check connection to verify the TFS material configuration

## Notes:

- Go will map TFS projects to destination folders within the Go agent installation directories. You can identify mappings by looking at the destination folders.
- In this release, Go does not delete any workspace it has created. Workspace names are generated internally.
- If possible, create a new user account on TFS which will be used for creating TFS materials in Go. You can use this account to easily identify TFS workspaces that Go created on the server and agents.
- If you are using the cross platform command line client, you cannot run the Go server and Go agent as a service on the local system account. You will need to run the service with a user account which has accepted the eula for the client.
- If at any point, you need to change the go server installation to a different location or machine, you will need to manually delete the old tfs mappings at the old Go server location.

## Known Caveats

- If TFS server is accessed using HTTPS and the SSL certificate is an untrusted certificate then the certificate must be added to the trust store of the Java installation used to launch the Go server and agents. Untrusted certificate will not be trusted by default Go server and agents.
- During the TFS checkout process, if one of the file paths exceeds 259 characters - checkout will fail. So care should be taken when specifying the destination directory so that the path limit is not exceeded. Go agent installation directory also plays a part in both cases where destination directory is specified and when its not.
- On all the agents prior to checkout from TFS, entire mapped directory is cleaned and -all option is used for checkout. As a result there is increased load on TFS server and network bandwidth consumption is high during the process. This is an issue that will be addressed in subsequent releases.
- Kerberos support for TFS authentication has not been verified.
- TFS 2011 support has not been verified.
- Currently, Go always does a tfs get to retrieve the latest changes on the agents.

# Go Configuration Reference

```
<cruise>
  <server>
    <license/>
    <security>
      <ldap/>
      <passwordFile/>
      <roles>
        <role>
          <user/>
        </role>
      </roles>
      <admins>
        <role/>
        <user/>
      </admins>
    </security>
    <mailhost/>
  </server>

  <repositories>
    <repository>
      <pluginConfiguration/>
      <configuration>
        <property>
          <key/>
          <value/>
        </property>
      </configuration>
      <packages>
        <package>
          <configuration>
            <property>
              <key/>
              <value/>
            </property>
          </configuration>
        </package>
      </packages>
    </repository>
  </repositories>

```

```
</repositories>

<pipelines>
    <authorization>
        <admins>
            <user/>
            <role/>
        </admins>
        <view>
            <user/>
            <role/>
        </view>
        <operate>
            <user/>
            <role/>
        </operate>
    </authorization>

    <pipeline>
        <params>
            <param/>
        </params>
        <trackingtool/>
        <mingle/>
        <timer/>
        <environmentvariables>
            <variable>
                <value/>
            </variable>
        </environmentvariables>

        <materials>
            <svn>
                <filter>
                    <ignore/>
                </filter>
            </svn>
            <hg>
                <filter>
                    <ignore/>
                </filter>
            </hg>
            <p4>
                <view/>
                <filter>
```

```
        <ignore/>
    </filter>
</p4>
<git>
    <filter>
        <ignore/>
    </filter>
</git>
<tfs>
    <filter>
        <ignore/>
    </filter>
</tfs>
<package/>
<pipeline/>
</materials>

<stage>
    <approval>
        <authorization>
            <role/>
            <user/>
        </authorization>
    </approval>
    <environmentvariables>
        <variable>
            <value/>
        </variable>
    </environmentvariables>

    <jobs>
        <job>
            <environmentvariables>
                <variable>
                    <value/>
                </variable>
            </environmentvariables>
            <resources>
                <resource/>
            </resources>

            <tasks>
                <fetchartifact>
                    <runif/>
                    <cancel/>
                </fetchartifact>
            </tasks>
        </job>
    </jobs>

```

```
        </fetchartifact>
        <ant>
            <runif/>
            <cancel/>
        </ant>
        <nant>
            <runif/>
            <cancel/>
        </nant>
        <rake>
            <runif/>
            <cancel/>
        </rake>
        <exec>
            <arg/>
            <runif/>
            <cancel/>
        </exec>
    </tasks>

    <artifacts>
        <artifact/>
        <test/>
    </artifacts>
    <tabs>
        <tab/>
    </tabs>
    <properties>
        <property/>
    </properties>
</job>
</jobs>
</stage>
</pipeline>
</pipelines>

<templates>
    <pipeline>
        <stage>
            ...
        </stage>
    </pipeline>
</templates>

<environments>
```

```
<environment>
    <environmentvariables>
        <variable>
            <value/>
        </variable>
    </environmentvariables>
    <agents>
        <physical/>
    </agents>
    <pipelines>
        <pipeline/>
    </pipelines>
</environment>
</environments>

<agents>
    <agent>
        <resources>
            <resource/>
        </resources>
    </agent>
</agents>
</cruise>
```

[top](#)

# Configuration reference

---

## `<cruise>`

---

The `<cruise>` element is the root element of the configuration.

[top](#)

## `<server>`

---

The `<server>` element can be used to define information and attributes of the Go Server.

## Attributes

---

Attribute	Required	Description
artifactsdir	No	<p>This directory is where Go will store information, including artifacts published by jobs. The <b>default value</b> 'artifacts' in the folder where the Go Server is installed. You can use an absolute path or a relative path which will take the server installed directory as the root. <b>Notes:</b> If you specify this attribute, please check whether Go has permission to access that directory. Also you should be aware of that changing this value while Go Server is running won't take effect until Go Server is restarted.</p>
siteUrl	No	<p>This entry will be used by Go Server to generate links for emails, feeds etc. where we cannot have relative URLs. For example, if you have fronted Go with a reverse proxy, this value should be the base URL for the proxy and not the internal Go address. For this reason, it is necessary to specify this configuration. Format: [protocol]://[host]:[port]. You need to define the [port] in case Go uses a non-standard port.</p>
secureSiteUrl	No	<p>Certain features in Go, such as Minion integration, require an HTTPS(SSL) endpoint. If you wish that your primary site URL be HTTP, but still want to have HTTPS endpoints for the features that require SSL, you can specify the secureSiteUrl attribute with a value of the base HTTPS URL. Format: <a href="https://[host]:[port]">https://[host]:[port]</a>. You need to define the [port] in case Go uses a non-standard port.</p>
purgeStart	No	<p>Go can purge old artifacts when available disk space on the server is low. Go will begin to purge artifacts when disk space is lower than 'purgeStart' GB. Artifacts will never be deleted if 'purgeStart' and 'purgeUpto' are not defined.</p>
purgeUpto	No	<p>Go can purge old artifacts when available disk space on the server is low. Go will purge artifacts till available disk space is greater than 'purgeUpto' GB. This attribute must be defined if 'purgeStart' is defined.</p>
jobTimeout	No	<p>This entry will be used as the default timeout value for hung jobs. A job is considered as hung if it does not generate any console output for "jobTimeout" minutes. If the attribute is not specified jobTimeout defaults to</p>

		minutes.
commandRepositoryLocation	Yes (auto-generated)	Specifies the location of the <a href="#">comma repository</a> relative to <code>go-server_install_root/db/command_repository</code> . The bundled repository is in a directory named default.
serverId	Yes (auto-generated)	This value uniquely identifies a Go server installation. It may be used by features that require unique names/identifiers across different Go server installations. This attribute needs to be specified for a new server. In case no value is given, server auto-generates a random UUID and assigns it as serverId. This value should never be changed for an existing server. Administrator should clear this attribute before copying configuration to a different installation.
agentAutoRegisterKey	No	The key specified here is used by agents for <a href="#">auto-registration</a> .

## Notes:

- If both siteUrl and secureSiteUrl are not defined, Go URLs will use the default domain which in most cases will be <http://your-go-server:8153>
- If only siteUrl is defined and is not HTTPS, Go URLs will be composed from the siteUrl entry. In this case, the secure pages of Go will not be navigable.
- If only siteUrl is defined and is HTTPS, Go URLs will be composed from the siteUrl entry and all pages will be HTTPS.
- If only secureSiteUrl is defined, Go URLs will use the default domain for non-HTTPS pages, while HTTPS pages will be composed from the secureSiteUrl entry.
- If purgeStart and purgeUpto are not defined, artifacts will never be deleted.

## Examples

```
<cruise>
  <server artifactsdir="/var/lib/go/big-artifacts-folder" siteUrl="http://go.example.com">
    <license user="${the user name in your license}">
      ${your license key}
    </license>
  </server>
</cruise>
```

## <license>

The `<license>` element contains your Go license. Go works in the community edition if this element is not present. To obtain a Go license, please visit the [Go Website](#).

## Attributes

Attribute	Required	Description
user	Yes	the user name in your

## Examples

```
<server artifactsdir="/var/lib/go/big-artifacts-folder">
  <license user="${the user name in your license}">
    ${license key}
  </license>
</server>
```

[top](#)

## <security>

The `<security>` element can be used to enable authentication. If the element is not defined anyone can use Go without logging in. We currently support [LDAP](#) and a simple [password file](#) format. You can use both methods if you want. This can be useful if you want to allow access from scripts without having to add a lot of users to your corporate LDAP. In this case you could add a 'script' user to the password file.

## Attributes

Attribute	Required	Description
allowOnlyKnownUsersToLogin	No	Allow only those users to login who have been explicitly added by an admin. If false, any new user who tries to login and is present in your password file or LDAP will be automatically created as a Go user. (Default=false)

# Examples

```
<server artifactsdir="/var/lib/go/big-artifacts-folder">
  <license user="${the user name in your license}">
    ${license key}
  </license>
  <security allowOnlyKnownUsersToLogin="false">
    <ldap uri="ldap://xxx.yourcompany.com"
      managerDn="cn=Acitivity Directory LDap User,ou=InformationSystems,ou=SharedAcc
      managerPassword="password"
      searchBase="ou=Employees,ou=Enterprise,ou=Principal,dc=xxxx,dc=yyyy,dc=com"
      searchFilter="(sAMAccountName={0})" />
  </security>
</server>
```

[top](#)

## <mailhost>

The `<mailhost>` element is used to configure mail notifications. Mail notifications require [security](#) to be enabled.

Attribute	Required	Description
hostname	Yes	The SMTP mail server which Go will use to send email. For example, <code>hostname="mailhost.yourcompany.com"</code>
port	Yes	The mail port to use. Typically this will be the default mail port of 25.
username	Yes	The username which Go should use to login to the mail host.
password	Yes	The password to access the mailhost.
tls	No	Use TLS(Transport Layer Security) or not. The default value is 'false'. Use 'true' to enable TLS security.
from	Yes	Go will attempt to set this email address as the 'from' address for email that it sends. Note that this setting may not be honoured by the SMTP server that you use. For example, <code>from="go-admin@yourcompany.com"</code> .
admin	Yes	Go administrator's email address. Go will send diagnostic messages to this email address. For example, Go will send a warning message if it is running out of disk space.

# Examples

```
<mailhost hostname="mailhost.yourcompany.com" port="25" username="go-user" password='
```

top

## <ldap>

The `<ldap>` element is used to specify the ldap server. Users can access Go with their username and password from this ldap server.

## Attributes

Attribute	Required	Description
uri	Yes	uri for the ldap server. For example, uri="ldap://
managerDn	Yes	For example, managerDn="cn=Active Directory User,ou=InformationSystems,ou=SharedAccou
managerPassword	Yes	Go will connect to the LDAP server with this passw
searchBase	Yes	e.g. searchBase="ou=Employees,ou=Enterprise
searchFilter	No	e.g. searchFilter="(sAMAccountName={0})"

# Examples

```
<security>
  <ldap uri="ldap://xxx.yourcompany.com"
        managerDn="cn=Acitivity Directory Ldap User,ou=InformationSystems,ou=SharedAccou
        managerPassword="password"
        searchBase="ou=Employees,ou=Enterprise,ou=Principal,dc=xxxx,dc=yyyy,dc=com"
        searchFilter=" (sAMAccountName={0})" />
  <passwordFile path="/home/go/admins.properties"/>
  <roles>
    <role name="go-admin">
      <user>Jez</user>
      <user>lqiao</user>
    </role>
  </roles>
  <admins>
    <role>go-admin</role>
    <user>lqiao</user>
  </admins>
</security>
```

[top](#)

## <passwordFile>

The `<passwordFile>` element is used to specify a file which has a set of username and password pairs. The format of username and password in this file is  `${username}=${password}` which has been encrypted with SHA1}, with one line per user.

Attribute	Required	Description
path	Yes	The absolute path of the password file.

## Examples

Suppose the password file is **admins.properties**, which is located in **/home/go**. You want to create two users as Administrators:

- one username is **Jez**, the password encrypted with SHA1 is **ThmbShxAtJepX80c2JY1FzOEmUk=**
- the other one is **Iqiao**, the password encrypted with SHA1 is **TfkgsHsIgJepX80c2JY1trwEskT=**

The configuration could look like:

```
<security>
  <passwordFile path="/home/go/admins.properties"/>
</security>
```

The username and password could be set in admins.properties as follows:

```
Jez=ThmbShxAtJepX80c2JY1FzOEmUk
Iqiao=TfkgsHsIgJepX80c2JY1trwEskT
```

[top](#)

## <roles>

The `<roles>` element is a container for roles that users defined. It can't be defined without `<role>`.

## Examples

```
<security>
  ...
<roles>
  <role name="go-admin">
    <user>Jez</user>
    <user>lqiao</user>
  </role>
</roles>
</security>
```

[top](#)

## <role>

The `<role>` element is used to define a group of users who perform similar tasks. Each user is added by adding the sub-tag `<user>`.

### Notes:

- If you want to define roles, you must define an authentication method, either `<ldap>` or `<passwordFile>`.
- These roles are not associated with roles defined in LDAP; they only work within Go. For example, you can assign a role to the `manual-approval` in a stage, so that only the users in that role can approve the stage to run.

Attribute	Required	Description
name	Yes	The name of the role.

## Examples

Two users would be in the role 'pipeline-operators', they are **Jez** and **Iqiao**.

```
<roles>
  <role name="pipeline-operators">
    <user>Jez</user>
    <user>lqiao</user>
  </role>
```

```
</roles>
```

[top](#)

## <user>

One `<user>` element defines a particular user in a role. You can add as many as you like.

### Notes:

- The user must be in your [LDAP](#) or [passwordFile](#).

## Examples

Two users would be in the role 'pipeline-operators', they are **Jez** and **Iqiao**.

```
<role name="pipeline-operators">
  <user>Jez</user>
  <user>lqiao</user>
</role>
```

[top](#)

## <admins>

The `<admins>` element specifies which users are administrators. Only administrators can open the Administration tab to maintain Go Configuration. Administrators can perform all functions in Go (including triggering pipelines, deploying to environments etc.)

### Notes:

The user must be in your [LDAP](#) or [passwordFile](#).

## Examples

```
<security>
  ...
  <admins>
    <role>go-admin</role>
```

```
<user>lqiao</user>
</admins>
</security>
```

[top](#)

## <role>

One `<role>` element in `<admins>` is used to specify a group as administrators. You can add as many as you like.

### Notes:

- The role must refer to `<roles>`.

## Examples

The users in role '**go-admin**' would be administrators.

```
<admins>
  <role>go-admin</role>
  <user>lqiao</user>
</admins>
```

[top](#)

## <user>

### Notes:

- The user must be in your [LDAP](#) or [passwordFile](#).

## Examples

Two users would be administrators, they are **Jez** and **Iqiao**.

```
<admins>
  <user>Jez</user>
  <user>lqiao</user>
</admins>
```

[top](#)

## <role>

### Notes:

- The role must be defined in `<roles>`.

## Examples

```
<view>
  <user>lqiao<user>
    <role>_readonly_member<role>
  </view>
```

[top](#)

## <repositories>

The `<repositories>` element is a container of package repositories.

## Example

```
<cruise>
  ...
  <repositories>
    <repository id="repo-id" name="repo-name">
      <pluginConfiguration id="plugin-id" version="plugin-version" />
      <configuration>
        <property>
          <key>property-name</key>
          <value>property-value</value>
        </property>
        ...
      </configuration>
      <packages>
        <package id="package-id" name="package-name" >
          <configuration>
            <property>
              <key>property-name</key>
              <value>property-value</value>
            </property>
            ...
          </configuration>
        </package>
      </packages>
    </repository>
  </repositories>
</cruise>
```

```

    </package>
  </packages>
</repository>
</repositories>
</cruise>

```

[top](#)

## <repository>

The `<repository>` element specifies a single repository. Repository must be unique by id and name (name is case-insensitive) across repositories configuration.

## Attributes

Attribute	Required	Description
id	No	The id uniquely identifies a package repository by GO. This attribute need not be specified. In case no value is given, server auto-generates a random UUID and assigns it as repository id.
name	Yes	The name uniquely identifies a package repository which will be specified by user and same will be used to display on screen. Repository name can contain the following characters: a-z, A-Z, 0-9, fullstop, underscore and hyphen. Spaces are not allowed. Name is case-insensitive in Go and the length should be less than 255 characters.

## Example

```

<cruise>
  ...
<repositories>
  <repository id="repo-id" name="repo-name">
    <pluginConfiguration id="plugin-id" version="plugin-version" />
    <configuration>
      <property>
        <key>property-name</key>
        <value>property-value</value>
      </property>
      ...
    </configuration>
  </repository>
</repositories>
<packages>
  <package id="package-id" name="package-name" >
    <configuration>
      <property>
        <key>property-name</key>
        <value>property-value</value>
      </property>
    </configuration>
  </package>
</packages>

```

```
</property>
...
</configuration>
</package>
</packages>
</repository>
</repositories>
</cruise>
```

[top](#)

## <pluginConfiguration>

The `<pluginConfiguration>` element specifies configuration related to plugin.

### Attributes

Attribute	Required	Description
id	Yes	Specifies plugin id which is going to handle repository configuration
version	Yes	Specifies plugin version which is going to handle repository configuration

[top](#)

## <configuration>

The `<configuration>` element specifies configuration related repository or package as one or more properties.

[top](#)

## <property>

The `<property>` element holds key and value.

[top](#)

## <key>

The `<key>` element specifies name of property.

[top](#)

## `<value>`

The `<value>` element specifies value of property.

[top](#)

## `<packages>`

The `<packages>` element specifies list of packages under a repository.

[top](#)

## `<package>`

The `<package>` element specifies single package under a repository. This tag holds configuration related to package

## Attributes

Attribute	Required	Description
id	No	The id uniquely identifies a package by GO across repositories. This attribute need not be specified. In case no value is given, server auto-generates a random UUID and assigns it as package id.
name	Yes	The name uniquely identifies a package within a repository, name will be specified by user and same will be used to display on screen. Package name can contain the following characters: a-z, A-Z, 0-9, fullstop, underscore and hyphen. Spaces are not allowed. Name is case-insensitive in Go and the length should be less than 255 characters.
autoUpdate	No	By default Go polls the repository for changes automatically. If autoUpdate is set to false then Go will not poll the repository for changes. Instead it will check for changes only when you trigger a pipeline that contains this material.

[top](#)

## <pipelines>

The `<pipelines>` element is a container of pipelines.

## Attributes

Attribute	Required	Description
group	No	The name is used to identify a pipeline group, and must be unique. The name can contain the following characters: a-z, A-Z, 0-9, period (.), underscore (_) and hyphen (-). Spaces are not allowed. The length should be less than 255 characters. The default name is 'defaultGroup'.

## Examples

```
<cruise>
  ...
  <pipelines group="studios">
    <pipeline name="yourproject" labeltemplate="foo-1.0.${COUNT}">
      <materials>
        <svn url="http://your-svn/" />
      </materials>
      <stage name="ut">
        <jobs>
          <job name="linux">
            <resources>
              <resource>linux</resource>
            </resources>
            <tasks>
              <ant target="unit-test" />
            </tasks>
          </job>
        </jobs>
      </stage>
    </pipeline>
  </pipelines>
</cruise>
```

[top](#)

## <authorization>

The `<authorization>` tag allows you to specify what users and roles are able to administer, view or operate any particular group of pipelines.

[top](#)

## <admins>

The `<admins>` element is a permission section to specify who can administer the pipeline group. Go administrators can define roles and users in the tag.

Users and Roles defined as group admins can view and operate on all pipelines in this pipeline group. They are allowed to navigate to the admin page where they can only see and edit this pipeline group which includes creating and modifying pipelines in this group, via the Pipeline Configuration Tab. They have no permission to view or modify the Pipeline Templates even if they are used by any pipeline in this group.

**Note:** Go Administrators (`admins`) defined in [security](#) tab, can administer all pipeline groups.

## Examples

Given the following configuration only `admins`, lqiao and any users having the role 'studios\_group\_admin'.

```
<cruise>
...
<pipelines group="studios">
  <authorization>
    <admins>
      <user>lqiao</user>
      <role>studios_group_admin</role>
    </admins>
  </authorization>
  <pipeline name="yourproject" labeltemplate="foo-1.0.{COUNT}">
    .....
  </pipeline>
</pipelines>
</cruise>
```

[top](#)

## <view>

The `<view>` element is a permission section to specify who can see the pipelines under the pipeline group. You can define roles and users in the tag.

**Note:** Administrators ([admins](#)) can see all pipeline groups. Any other users or roles that are not listed under the `<view>` tag will be unable to see this pipeline group

## Examples

Given the following configuration only administrators can operate the pipeline group, and only [admins](#), lqiao and any users having the role 'go\_READONLY\_member' can see the pipeline.

```
<cruise>
...
<pipelines group="studios">
  <authorization>
    <view>
      <user>lqiao</user>
      <role>go_READONLY_member</role>
    </view>
  </authorization>
  <pipeline name="yourproject" labeltemplate="foo-1.0.${COUNT}">
    .....
  </pipeline>
</pipelines>
</cruise>
```

[top](#)

## <operate>

The `<operate>` element specifies who can operate the pipelines under the pipeline group. You can define roles and users.

**Note:** Any users/roles that are not listed under the `<view>` tag will be unable to see this pipeline group (even if they are listed as being allowed to `<operate>` that pipeline group)

## Examples

Given the following configuration, only [admins](#), lqiao, jez and the users having the role 'go\_core\_member' can operate the pipeline group. Only [admins](#), lqiao and the users having the role 'go\_READONLY\_member' can see the pipeline (jez and go\_core\_member cannot).

```
<cruise>
...

```

```

<pipelines group="studios">
  <authorization>
    <view>
      <user>lqiao</user>
      <role>go_READONLY_member</role>
    </view>
    <operate>
      <user>lqiao</user>
      <user>jez</user>
      <role>go_CORE_member</role>
    </operate>
  </authorization>
  <pipeline name="yourproject" labeltemplate="foo-1.0.${COUNT}">
    .....
  </pipeline>
</pipelines>
</cruise>

```

[top](#)

## <pipeline>

The `<pipeline>` element specifies a single pipeline. It must be unique (including case) across the entire configuration (not only in the pipeline group).

### Notes:

There should be at least one stage in one pipeline. Go uses the pipeline name to identify the pipeline. If you change the pipeline name, you will lose the history of the pipeline.

## Attributes

Attribute	Required	Description
name	Yes	The name is used to identify a pipeline, so each pipeline name must be unique. Pipeline name can contain the following characters: a-z, A-Z, 0-9, fullstop, underscore and hyphen. Spaces are not allowed. Name is case-insensitive in Go and the length should be less than 255 characters.
labeltemplate	No	Both of material names and \${COUNT} are available in the labeltemplate and the default value of labeltemplate is '\${COUNT}'. If you just specify labeltemplate="foo-1.0-\${COUNT}", your pipeline will show foo-1.0-1, foo-1.0-2, and so on. When you reference material names in the labeltemplate, Go will use the revisions of the reference materials to populate the pipeline label. For example, given a material name is 'svnrepo' in a pipeline, when you specify labeltemplate="foo-

		1.0-\${svnrepo}", then your pipeline would show foo-1.0-3123, foo-1.0-3124, and so on. Material names are case insensitive. The max length of a pipeline label is 255. If a material name is 'svnrepo', the following labeltemplates are valid: \${COUNT}, \${svnrepo}, foo-\${COUNT}-\${SVNrepo}, foo-\${svnrepo}-\${COUNT}-bar.
isLocked	No	The possible values are "true" or "false". The default value is "false". When set to "true" Go ensures that only a single instance of a pipeline can be run at a time.
template	No	The name of the template that this pipeline references. If this is set, no stages may be defined in this pipeline.

## Examples

```

<pipelines>
  <pipeline name="yourproject" labeltemplate="foo-1.0.${COUNT}-${svn}" isLocked="true">
    <environmentvariables>
      <variable name="FOO"><value>bar</value></variable>
    </environmentvariables>
    <materials>
      <svn url="http://your-svn/" materialName="svn" />
    </materials>
    <stage name="ut">
      <jobs>
        <job name="linux">
          <resources>
            <resource>linux</resource>
          </resources>
          <tasks>
            <ant target="unit-test" />
          </tasks>
        </job>
      </jobs>
    </stage>
  </pipeline>
</pipelines>

```

[top](#)

## <params>

The element `<params>` specifies the list of parameters (Element `param`) elements to be used in a pipeline or a pipeline template. You can specify these under a `<pipeline>` and can be used anywhere inside pipeline/stage/job definition.

[top](#)

## <param>

A `<param>` defines the parameter name that will be substituted with the parameter value that will be substituted in a pipeline or a pipeline template.

### Example:

```
<params>
  <param name="COMMAND">echo</param>
  <param name="WORKING_DIR">/repo/branch</param>
</params>
```

[top](#)

## <trackingtool>

The `<trackingtool>` element can be used to specify links to an issue tracker. Go will construct a link based on the commit message that you can use to take you to your tracking tool (Mingle card, JIRA issue, Trac issue etc).

### Attributes

Attribute	Required	Description
link	Yes	a URL with a string ' <code>#{ID}</code> '. Go will replace the string ' <code>#{ID}</code> ' with the first matched group value at run-time.
regex	Yes	A <a href="#">regex</a> to identify the IDs. Go will find the first matched group in your commit messages and use it to construct the hyper-link.

### Examples

Suppose you are using a Web Application to manage your tasks or bugs, and the link looks like <http://your-trackingtool/yourproject/512>, '512' is your task ID. Your configuration would be:

```
<pipeline name="yourproject">
  <trackingtool link="http://your-trackingtool/yourproject/${ID}" regex="evo-(\d+)" />
```

```
...  
</pipeline>
```

If you check in some code with a commit message which includes the characters 'evo-512' then that will appear in the modification pop-up box as a link. When you click it, Go will take you to the web page '<http://your-trackingtool/yourproject/512>'.

For example: If you use [Mingle](#) for your task manager, the configuration would be:

```
<pipeline name="yourproject">  
  <trackingtool link="http://your-mingle-server/projects/yourproject/cards/${ID}" rec  
  ...  
</pipeline>
```

**Notes:** You can not define multiple tracking tools in one pipeline.

[top](#)

## **<mingle>**

This element let's you associate a [Mingle](#) project to a pipeline. Once associated, you will be able to track Mingle cards from within Go.

**Note:** You cannot configure a [trackingtool](#) if mingle is configured for a pipeline.

## **Attributes**

Attribute	Required	Description
baseUrl	Yes	Base URL to the Mingle installation (do not include the project name/identifier)
projectIdIdentifier	Yes	This is the "Identifier" specified under a Mingle project's "Basic Options"
mqlGroupingConditions	No	An MQL string that determines the "passing criteria" for cards displayed in Go

## **Examples**

```

<mingle
    baseUrl="http://mingle.example.com"
    projectIdentifier="my_project">
    <mqlGroupingConditions>status > 'In Dev'</mqlGroupingConditions>
</mingle>

```

[top](#)

## <timer>

The `<timer>` element specifies a cron-like schedule to build the pipeline.

## Attributes

Attribute	Required	Description
onlyOnChanges	No	Skips scheduling if the previous run of the pipeline was with the latest material(s). This option is typically useful when automatic pipeline scheduling is turned off.

## Examples

For example to run a pipeline once a night at 10pm on weekdays:

```

<pipeline name="yourproject">
    <timer>0 0 22 ? * MON-FRI</timer>
    ...
</pipeline>

```

Go uses the [Quartz](#) scheduler internally. For convenience we reproduce the [Quartz cron documentation](#) here:

## Format

A cron expression is a string comprised of 6 or 7 fields separated by white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

Field Name	Mandatory?	Allowed Values	Allowed Special Characters
Seconds	YES	0-59	, - * /

Minutes	YES	0-59	, - * /
Hours	YES	0-23	, - * /
Day of month	YES	1-31	, - * ? / L W\
Month	YES	1-12 or JAN-DEC	, - * /
Day of week	YES	1-7 or SUN-SAT	, - * ? / L #
Year	NO	empty, 1970-2099	, - * /

So cron expressions can be as simple as this: `* * * * *` or more complex, like this: `0 15 10 ? * 6L 2002-2005`

## Special characters

- `*` ("all values") - used to select all values within a field. For example, "\*" in the minute field means "every minute".
- `?` ("no specific value") - useful when you need to specify something in one of the two fields in which the character is allowed, but not the other. For example, if I want my trigger to fire on a particular day of the month (say, the 10th), but don't care what day of the week that happens to be, I would put "10" in the day-of-month field, and "?" in the day-of-week field. See the examples below for clarification.
- `-` - used to specify ranges. For example, "10-12" in the hour field means "the hours 10, 11 and 12".
- `,` - used to specify additional values. For example, "MON,WED,FRI" in the day-of-week field means "the days Monday, Wednesday, and Friday".
- `/` - used to specify increments. For example, "0/15" in the seconds field means "the seconds 0, 15, 30, and 45". And "5/15" in the seconds field means "the seconds 5, 20, 35, and 50". You can also specify '/' after the "**character - in this case**" is equivalent to having '0' before the '/'. '1/3' in the day-of-month field means "fire every 3 days starting on the first day of the month".
- `L` ("last") - has different meaning in each of the two fields in which it is allowed. For example, the value "L" in the day-of-month field means "the last day of the month" - day 31 for January, day 28 for February on non-leap years. If used in the day-of-

week field by itself, it simply means "7" or "SAT". But if used in the day-of-week field after another value, it means "*the last xxx day of the month*" - for example "6L" means "*the last friday of the month*". When using the 'L' option, it is important not to specify lists, or ranges of values, as you'll get confusing results.

- `w` ("weekday") - used to specify the weekday (Monday-Friday) nearest the given day. As an example, if you were to specify "15W" as the value for the day-of-month field, the meaning is: "*the nearest weekday to the 15th of the month*". So if the 15th is a Saturday, the trigger will fire on Friday the 14th. If the 15th is a Sunday, the trigger will fire on Monday the 16th. If the 15th is a Tuesday, then it will fire on Tuesday the 15th. However if you specify "1W" as the value for day-of-month, and the 1st is a Saturday, the trigger will fire on Monday the 3rd, as it will not 'jump' over the boundary of a month's days. The 'W' character can only be specified when the day-of-month is a single day, not a range or list of days. The 'L' and 'W' characters can also be combined in the day-of-month field to yield 'LW', which translates to "*last weekday of the month*".
- `#` - used to specify "the nth" XXX day of the month. For example, the value of "6#3" in the day-of-week field means "*the third Friday of the month*" (day 6 = Friday and "#3" = the 3rd one in the month). Other examples: "2#1" = the first Monday of the month and "4#5" = the fifth Wednesday of the month. Note that if you specify "#5" and there is not 5 of the given day-of-week in the month, then no firing will occur that month. The legal characters and the names of months and days of the week are not case sensitive. `MON` is the same as `mon`.

## Examples

Here are some full examples:

Expression	Meaning
<code>0 0 12 * *</code> ?	Fire at 12pm (noon) every day
<code>0 15 10 ? *</code> *	Fire at 10:15am every day
<code>0 15 10 * *</code> ?	Fire at 10:15am every day
<code>0 15 10 * *</code> ? *	Fire at 10:15am every day
<code>0 15 10 * *</code> ? 2005	Fire at 10:15am every day during the year 2005
<code>0 * 14 * *</code>	Fire every minute starting at 2pm and ending at 2:59pm, every

?	day
0 0/5 14 * *	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day
* * ? 0 0/5 14,18	Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire every 5 minutes starting at 6pm and ending at 6:55pm, every day
? 0 0-5 14 * *	Fire every minute starting at 2pm and ending at 2:05pm, every day
0 10,44 14 ? 3 WED	Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
0 15 10 ? * MON-FRI	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday
0 15 10 15 *	Fire at 10:15am on the 15th day of every month
0 15 10 L *	Fire at 10:15am on the last day of every month
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month
0 15 10 ? * 6L 2002-2005	Fire at 10:15am on every last friday of every month during the years 2002, 2003, 2004 and 2005
0 15 10 ? * 6#3	Fire at 10:15am on the third Friday of every month
0 0 12 1/5 *	Fire at 12pm (noon) every 5 days every month, starting on the first day of the month.
0 11 11 11 11 ?	Fire every November 11th at 11:11am.

Pay attention to the effects of '?' and '\*' in the day-of-week and day-of-month fields!

## Notes

- Support for specifying both a day-of-week and a day-of-month value is not complete (you must currently use the '?' character in one of these fields).
- Be careful when setting fire times between mid-night and 1:00 AM - "daylight savings" can cause a skip or a repeat depending on whether the time moves back or jumps forward.

[top](#)

**<materials>**

---

The `<materials>` element specifies the source of the pipeline changes. Generally this will be your codebase in your source control repository.

## Notes:

Go supports multiple materials with the restriction that every material must contain a unique "dest" folder (that is not a subfolder of any other material). Go will check out the source code into this folder for each material.

## Examples

```
<pipeline name="yourproject" labeltemplate="foo-1.0.${COUNT}">
  <materials>
    <svn url="http://your-svn/" />
  </materials>
  ...
</pipeline>
```

### Multiple materials:

```
<pipeline name="yourproject" labeltemplate="foo-1.0.${COUNT}">
  <materials>
    <svn url="http://your-svn/" dest="svn-folder" />
    <git url="git://127.0.0.1/precommit.git" branch="1.3branch" dest="git-folder" />
    <hg url="http://your-hg/" dest="hg-folder" />
    <p4 port="10.18.3.102:1666" username="userName" password="passwd" dest="p4-folder">
      <view><![CDATA[
        //depot/dev/src...
      ]]></view>
    </p4>
  </materials>
  ...
</pipeline>
```

[top](#)

## `<filter>`

---

The `<filter>` element specifies files in changesets that should not trigger a pipeline automatically. When a pipeline is triggered by files that are not ignored the filtered files will still be updated with other files. You can only define one filter under each SCM

material. When you trigger a pipeline manually, it will update to most recent revision, including filtered files.

## Examples

```
<svn url="http://your-svn/">
  <filter>
    <ignore pattern="doc/**/*.*" />
  </filter>
</svn>
```

[top](#)

### <ignore>

The `<ignore>` element is used to specify a set of files that are ignored when Go checks for changes. Repository changesets which only contain these files will not trigger a pipeline automatically.

## Attributes

Attribute	Required	Description
pattern	Yes	defines a pattern (Ant-style) for the files to be ignored. Changes of those files will not trigger the pipeline. the pattern is relative to the root of the SCM repository, not the sandbox of the pipeline.

## Notes

- `<ignore>` can occur multiple times under `<filter>`.
- The pattern is relative to the root directory of the SCM repository, not the sandbox in the agent side or the materials URL.
- Ignored files are still updated when other files are updated.

## Examples:

```
<ignore pattern="doc/**/*" />
```

Ignore everything under the folder 'doc'.

```
<ignore pattern="doc/*" />
```

Ignore files under the folder '**doc**', excluding any subfolder.

```
<ignore pattern="framework/helper/*.doc" />
```

Ignore files that are under the directory 'framework/helper' and the file extension is **.doc**.

```
<ignore pattern="*.pdf" />
```

Ignore files that are under the root directory of SCM repository and the file extension is **.pdf**.

```
<ignore pattern="**/helper/*.pdf" />
```

Ignore all the files that is under any '**helper**' folder and the file extension is **.pdf**.

```
<ignore pattern="helper/**/*.pdf" />
```

Ignore all the files that are in the nested directory under folder '**helper**' of the repository and the file extension is **.pdf**.

[top](#)

## <svn>

The `<svn>` element specifies the location of your code base in Subversion repository.

## Attributes

Attribute	Required	Description
url	Yes	URL for the remote repository, for example: ' <a href="http://www.thoughtworks-studios.com/go-agile-release-management/svn/myproject/trunk">http://www.thoughtworks-studios.com/go-agile-release-management/svn/myproject/trunk</a> '. Go

		supports the following protocols for subversion: http, https, svn and svn+ssh, but does not support 'file:///'.
username	No	The user account for the remote repository.
password	No	The password for the specified user
checkexternals	No	The default value is false, the value should be either one of true/false or 1/0. 'true' or '1' means that the changes of externals will trigger the pipeline automatically.
dest	Required if there are multiple materials	The directory where the code will be checked out. This is relative to the sandbox of the Go Agent. Go prevents the destination folder from being outside the agent's sandbox.
materialName	Required if this material is referenced in pipeline labelTemplate	The name to identify a material. Material name can contain the following characters: a-z, A-Z, 0-9, fullstop, underscore and hyphen, but whitespace is not allowed. A material name is case insensitive and starting with fullstop is invalid. It needs to be unique within a pipeline. The max length is 255 characters.
autoUpdate	No	By default Go polls the repository for changes automatically. If autoUpdate is set to false then Go will not poll the repository for changes. Instead it will check for changes only when you trigger a pipeline that contains this material or it receives a notification through a post-commit hook. If the same material is specified more than once in the configuration file, all of them must have the same value for autoUpdate.

## Notes:

Go cannot automatically accept svn SSL certificates. If you are using https for svn repository, you have to go to the Server and each Agent, and as the user 'go' do a command "svn update" to store the certificates in the cache permanently.

## Examples:

For a Go Agent on linux with the following configuration:

```

<pipeline name="myproduct">
  <materials>
    <svn url="http://svn-server.com/framework" dest="framework"/>
    <svn url="http://svn-server.com/componentOne" dest="mycomponent"/>
  </materials>
  ...

```

```
</pipeline>
```

Go Agent will check out source code from '<http://svn-server.com/framework>' to '/var/lib/go-agent/pipelines/myproduct/framework', and from '<http://svn-server.com/componentOne>' to '/var/lib/go-agent/pipelines/myproduct/mycomponent'.

top

## <hg>

The `<hg>` element specifies the location of your code base in a Mercurial repository. Go supports the http and ssh for mercural.

### Notes:

You must install Mercurial 1.5 or above on the Go Server and Go Agents for the jobs need Mercurial. Go does not ship with Mercurial.

## Attributes

Attribute	Required	Description
url	Yes	URL to fetch source code from the Mercurial repository. If you specify the username and password for the Mercurial repository, you should put them into the url. Mercurial supports an optional identifier after # in the url, which indicates a particular branch, tag or changeset. This option can be used to configure mercurial branches in Go.
dest	Only for multiple materials	The directory where the code will be checked out. This is relative to the sandbox of the Go Agent. Go prevents the destination folder from being outside the agent's sandbox.
materialName	Required if this material is referenced in pipeline labeltemplate	The name to identify a material. Material name can contain the following characters: a-z, A-Z, 0-9, fullstop, underscore and hyphen. Spaces are not allowed. Material name is case insensitive. It needs to be unique within a pipeline. The max length is 255 characters.
autoUpdate	No	By default Go polls the repository for changes automatically. If autoUpdate is set to false then Go will not poll the repository for changes. Instead it will check for changes only when you trigger a pipeline that contains this material. If the same material is specified more than once in the configuration file, all of

them must have the same value for autoUpdate.

## Examples

```
<pipeline name="yourproject">
  <materials>
    <hg url="http://username:password@your-hg/" />
  </materials>
  ...
</pipeline>
```

## Specifying a mercurial branch.

```
<pipeline name="yourproject_branch">
  <materials>
    <hg url="http://username:password@your-hg##branch_name" />
  </materials>
  ...
</pipeline>
```

Note that # needs to be escaped with another # - hence the ## in the url above.

[top](#)

### <p4>

The `<p4>` element specifies the location of your code base in a Perforce repository.

## Notes:

Go will use directory under pipelines/{pipelineName} in agent side as Perforce root directory of perforce client workspace.

## Attributes

Attribute	Required	Description
port	Yes	Perforce server connection to use (host:port). This would pass in the p4port parameter for the p4 cc P4PORT environment variable.
username	No	Perforce username to use.

password	No	Password for the specified user.
useTickets	No	Set to true to work with perforce tickets. Go will check the supplied password before each command. We recommend you make your user a part of a p4 group, and set the unlimited as described here: <a href="http://www.perforce.com/perforce/doc.current/manuals/p4.go/config.html#useTickets">http://www.perforce.com/perforce/doc.current/manuals/p4.go/config.html#useTickets</a>
dest	Only for multiple materials	The directory where the code will be checked out from the sandbox of the Go Agent. Go prevents the destination outside the agent's sandbox.
view	Yes	Valid Perforce view. The view should be a sub-expression to see details about VIEW of Perforce.
materialName	Required if this material is referenced in pipeline labelTemplate	The name to identify a material. Material name can contain characters: a-z, A-Z, 0-9, fullstop, underscore and dash are not allowed. Material name is case insensitive. It is used within a pipeline. The max length is 255 characters.
autoUpdate	No	By default Go polls the repository for changes at regular intervals. autoUpdate is set to false then Go will not poll the repository for changes. Instead it will check for changes only when a pipeline that contains this material. If the same material appears more than once in the configuration file, all of the pipelines will have the value for autoUpdate.

## Notes:

You do not need to specify the above attributes if you have already defined them as system variables. So if you have a P4PASSWD variable defined then you can leave out the "password" tag defined above. If you already have them defined as system variables and also in Go configuration, Go will overwrite them before running p4.

Views consist of multiple mappings. Each mapping has two parts:

1. The left-hand side specifies one or more files in the depot and has the form:  
//depotname/file\_specification
2. The right-hand side specifies one or more files in the client workspace and has the form: //clientname/file\_specification

Go creates a p4 client to check out files into its sandbox with the 'clobber' option set. This means, during material update all writable-but-unopened files in the workspace would be overwritten on the agent. All other options use default values as defined by Perforce. Client name is generated automatically by Go. Hence, you can use anything as 'clientname' on the right-hand side in view mapping. The client name format is: cruise-[hostname]-[pipeline name]-[a random hash code], for example "cruise-myhostname-mypipelinename-wOaJ9kjpfOLQCncki19ikXt5Q". THE GO\_P4\_CLIENT

environment variable will have the client name used. This variable can be used in scripts to get the client name

Go views are in the same format as that used by Perforce itself. In fact you should be able to copy a Perforce view from your existing Perforce setup and paste it into the view section.

For example:

```
<pipeline name="perforce-example">
  <materials>
    <p4 port="10.18.3.102:1666" username="userName" password="passwd">
      <view><! [CDATA[
        //depot/dev/src...          //anything/src/...
        //depot/dev/test...         //anything/test/...
        //depot/dev/main/docs/...   //anything/docs/...
      ]]></view>
    </p4>
  </materials>
  ...
</pipeline>
```

[top](#)

## <git>

---

The `<git>` element specifies the location of your code base in a GIT repository. Go only supports remote repositories.

### Notes:

git versions 1.7 and above are supported by Go.

If 'branch' is defined, Go will check out the specified branch. Otherwise, Go will check out the master branch.

If there are submodules in the repository, Go will check out them as well.

msysGit on Windows has a [defect](#) which causes an error when using Go. Please ensure to use a build which fixes this.

While installing msysGit On Windows machines for Go server or agents, please choose Option iii, namely *Run Git and included UNIX tools from windows command prompt*

If you are using git through SSH on windows, please ensure that the HOME user environment variable is set to the full path of the parent directory where the .ssh/ directory is located.

## Attributes

Attribute	Required	Description
url	Yes	GIT URL for the repository.
branch	No	a branch name in the repository.
dest	Only for multiple materials	The directory under the sandbox of Go Agent. Go will check out the source code into this directory.
materialName	Required if this material is referenced in pipeline labeltemplate	The name to identify a material. Material name can contain the following characters: a-z, A-Z, 0-9, fullstop, underscore and hyphen. Spaces are not allowed. Material name is case insensitive. It needs to be unique within a pipeline. The max length is 255 characters.
autoUpdate	No	By default Go polls the repository for changes automatically. If autoUpdate is set to false then Go will not poll the repository for changes. Instead it will check for changes only when you trigger a pipeline that contains this material. If the same material is specified more than once in the configuration file, all of them must have the same value for autoUpdate.

## Examples are:

```
<pipeline name="yourproject">
  <materials>
    <git url="git://127.0.0.1/precommit.git" branch="1.3branch"/>
  </materials>
  ...
</pipeline>
```

```
<pipeline name="yourproject">
  <materials>
    <git url="http://ccgegit:pst@goserver.yourcompany.com/httpgit.git" />
  </materials>
  ...
</pipeline>
```

## <tfs>

The `<tfs>` element specifies the location of your code base in a TFS Source repository.

## Attributes

Attribute	Required	Description
URL	Yes	URL for the Collection on the TFS Server.
Domain	No	Domain name for TFS authentication credentials.
Username	Yes	Username of the account to access the TFS collection.
Password	Yes	Password of the account to access the TFS collection.
Project Path	Yes	The project path within the TFS collection.
dest	Only for multiple materials	The directory where the code will be checked out. This is relative to the sandbox of the Go Agent. Go prevents the destination folder from being outside the agent's sandbox.

## Examples are:

```
<pipeline name="webproject">
  <materials>
    <tfs url="http://10.21.3.210:8080/tfs/New" domain="DOMAIN" username="jim" password="jim" dest="wwwroot">
    </tfs>
  ...
</pipeline>
```

```
<pipeline name="myproject">
  <materials>
    <tfs url="http://tfshost.tw.com:8080/tfs/DefaultCollection" domain="DOMAIN" username="jim" password="jim" dest="wwwroot">
    </tfs>
  ...
</pipeline>
```

# <package>

The `<package>` element refers to package which is defined as part of repositories configuration.

## Attributes

Attribute	Required	Description
ref	Yes	The ref tag holds the id of the package

## Example

```
<cruise>
  ...
  <repositories>
    <repository id="repo-id" name="repo-name">
      <pluginConfiguration id="plugin-id" version="plugin-version" />
      <configuration>
        <property>
          <key>property-name</key>
          <value>property-value</value>
        </property>
        ...
      </configuration>
    </repository>
    <packages>
      <package id="1234-12242-232312" name="sample-package" >
        <configuration>
          <property>
            <key>property-name</key>
            <value>property-value</value>
          </property>
          ...
        </configuration>
      </package>
    </packages>
  </repositories>
  <pipelines name="webproject">
    ...
    <pipeline name="webproject">
      <materials>
        <package ref="1234-12242-232312" />
      </materials>
      ...
    </pipeline>
    ...
  </pipelines>
</cruise>
```

## <pipeline>

The `<pipeline>` element specifies that successful completion of a stage in another pipeline will trigger the current pipeline to start.

If there are multiple pipeline dependencies, then any one of them passing will trigger a new pipeline.

Note that you can not specify two (or more) dependencies for the same upstream pipeline.

## Attributes

Attribute	Required	Description
pipelineName	Yes	The name of a pipeline that this pipeline depends on.
stageName	Yes	The name of a stage which will trigger this pipeline once it is successful.
materialName	By default the materialName is the name of the upstream pipeline (the pipelineName). This is required if this material is referenced in pipeline labelTemplate	The name to identify a material. Material name can contain the following characters: a-z, A-Z, 0-9, fullstop, underscore and hyphen. Spaces are not allowed. Material name is case insensitive. It needs to be unique within a pipeline. The max length is 255 characters.

## Notes:

The downstream pipeline wouldn't be triggered if there was no passed stage in the upstream pipeline.

## Examples

Suppose there are four pipelines, and they are commonLib1, commonLib2, Server and Client. For example, the stage 'distStage' in commonLib1 pipeline can trigger the other two pipelines, and the stage 'pkgstage' in commonLib2 pipeline can trigger Server pipeline. The configuration would be:

```

<pipeline name="Server">
  <materials>
    <pipeline pipelineName="commonLib1" stageName="distStage"/>
    <pipeline pipelineName="commonLib2" stageName="pkgStage"/>
  </materials>
  ...
</pipeline>
<pipeline name="Client">
  <materials>
    <pipeline pipelineName="commonLib1" stageName="distStage"/>
  </materials>
  ...
</pipeline>

```

[top](#)

## <stage>

The `<stage>` element specifies a set of jobs. If any job in a given stage fails then the stage will fail. If a stage has an `<approval>` configuration with manual type it can only be triggered manually (i.e. a user must click on the trigger button on the UI). If the previous stage has failed, you can still trigger the following stage manually.

### Notes:

There must be at least one job in stage.

## Attributes

Attribute	Required	Description
name	Yes	The name is used to identify a stage in the pipeline, so it has to be unique (case insensitive) for that <code>&lt;pipeline&gt;</code> . The available characters in stage name are following: a-z, A-Z, 0-9, fullstop, underscore and hyphen. Spaces are not allowed.
fetchMaterials	No (Default: true)	Perform material updates or checkouts. Set this attribute to false to skip this operation.
cleanWorkingDir	No (Default: false)	Remove all files/directories in the working directory on the agent. By default this operation is skipped.
artifactCleanupProhibited	No (Default: false)	Never cleanup artifacts for this stage, if purging artifacts is configured at the Server Level.

# Examples

```
<pipeline name="yourproject">
  ...
  <stage name="ut">
    <environmentvariables>
      <variable name="FOO"><value>bar</value></variable>
    </environmentvariables>
    <jobs>
      <job name="linux">
        <resources>
          <resource>linux</resource>
        </resources>
        <tasks>
          <ant target="unit-test" />
        </tasks>
      </job>
    </jobs>
  </stage>
</pipeline>
```

[top](#)

## <jobs>

The `<jobs>` element specify the set of jobs for a stage.

### Note:

`<jobs>` can contain several `<job>` elements. These jobs can run in parallel on different `<agents>`.

# Examples

```
<stage name="ut">
  <jobs>
    <job name="linux">
      <resources>
        <resource>linux</resource>
      </resources>
      <tasks>
        <ant target="unit-test" />
      </tasks>
    </job>
  </jobs>
</stage>
```

## <job>

---

A job is the basic unit of work. It is executed on an agent. A job can fetch artifacts from Go Server, execute tasks and publish artifacts back to Go Server.

A job can also be associated with a set of <[resources](#)>. Resources are used to match a Job to an Agent. An Agent can run a Job if it has all the resources that the Job specifies.

If a Job has no resources then it can be built by any Agent

## Attributes

Attribute	Required	Description
name	Yes	The name of the job. The name must be unique (ignoring case) within a < <a href="#">stage</a> >. The name can contain: a-z, A-Z, 0-9, fullstop, underscore and hyphen only. Spaces are not allowed.
runOnAllAgents	No	If set to 'true' then the Job will run on all agents that can run the job.
runInstanceCount	No	If set to 'x' (integer) then 'x' instances of Job will be spawned during scheduling. Environment variables <code>GO_JOB_RUN_INDEX</code> (with values 1-x for every Job) and <code>GO_JOB_RUN_COUNT</code> (with value x for each Job) will be exposed to each task of Job.
timeout	No	A job can be configured to time out if it does not generate any console output for a period of time. Use this attribute to define the timeout value in minutes. Define timeout as 0 if the job should never time out. If the attribute is not defined, the default < <a href="#">server</a> > level timeout behaviour will apply.

## Examples

```
<job name="linux">
  <environmentvariables>
    <variable name="FOO"><value>bar</value></variable>
  </environmentvariables>
  <resources>
    <resource>linux</resource>
  </resources>
  <tasks>
```

```
<ant target="unit-test" />
</tasks>
</job>
```

```
<job name="run-upgrade" runOnAllAgents="true" timeout='30'>
  <resources>
    <resource>linux</resource>
  </resources>
  <tasks>
    <ant target="upgrade" />
  </tasks>
</job>
```

```
<job name="run-upgrade" runInstanceCount="5" timeout='30'>
  <resources>
    <resource>linux</resource>
  </resources>
  <tasks>
    <ant target="upgrade" />
  </tasks>
</job>
```

[top](#)

## **<resources>**

`<resources>` specifies the **resources** needed for a job. A job can have zero or more resources.

If a job has no resources it can be built on any agent.

## **Example:**

```
<job name="linux">
  <resources>
    <resource>jdk5</resource>
    <resource>tomcat5</resource>
    <resource>mercurial</resource>
  </resources>
</job>
```

[top](#)

## <resource>

---

A <resource> is a text tag that specifies a resource which a job requires to build. An Agent must have all the Resources specified for a Job to be able to run that Job.

## Validations:

Resources are case-insensitive. A resource name can contain alphanumeric characters, hyphens (-), spaces, periods (.) and pipes (|).

## Example:

```
<resources>
  <resource>jdk5</resource>
  <resource>tomcat5</resource>
  <resource>mercurial</resource>
</resources>
```

[top](#)

## <tasks>

---

<tasks> specifies the tasks (like [ant](#) , [rake](#) etc) that will run as part of a job.

There can be zero or more tasks. These tasks are executed in the order specified in the configuration file. If a task fails, the subsequent tasks are not run unless they have [`<runif status="failed" />`](#) defined.

The following environment variables are set for all tasks:

Attribute	Description
<code>GO_SERVER_URL</code>	The base URL for the server, includin <a href="https://localhost:8154/go">https://localhost:8154/go</a>
<code>GO_PIPELINE_NAME</code>	The name of the pipeline to which the
<code>GO_PIPELINE_LABEL</code>	The label of the pipeline to which the
<code>GO_STAGE_NAME</code>	The name of the stage to which the jo
<code>GO_STAGE_COUNTER</code>	The re-run counter of the stage to whi
<code>GO_JOB_NAME</code>	The name of the job that is being run

```
GO_DEPENDENCY_LABEL_
<upstream_pipeline_name>_<upstream_stage_name>
```

The label of the upstream pipeline which the job belongs to. For example 'GO\_DEPENDENCY\_LABEL\_FRAMEWORK' environment variable where the name pipeline is 'framework' and the upstream stage 'Hyphen ('-') is an illegal character in a pipeline name. So if a pipeline name or stage name contains a hyphen it is converted into an underscore. For example stage 'stage-foo' becomes: GO\_DEPENDENCY\_LABEL\_PIPELINEFOO

## Examples

```
<job name="linux">
  <tasks>
    <ant target="unit-test" />
  </tasks>
</job>
```

top

### <ant>

Specifies an Ant build to run. Ant is assumed to be present from the command line on the agent. Go depends on and uses JDK 1.6. If JDK 1.4 or 1.5 binaries are required by a build, it can be specified in the Ant [javac](#) task.

All paths specified are relative to the pipeline working directory.

## Attributes

Attribute	Required	Description
buildfile	No	Path to Ant build file. If not specified, the path defaults to 'build.xml'.
target	No	Ant target(s) to run. If not specified, the target defaults to 'default'
workingdir	No	The directory from where Ant is invoked

## Examples

- Invoke Ant, specifying a set of targets to run:

```
<tasks>
  <ant target="-Drun=all clean.ivy.localivy clean ft.long_running"/>
</tasks>
```

- Invoke Ant in a specific working directory with a set of targets:

```
<tasks>
  <ant workingdir="build" buildfile="mybuild.xml" target="-Drun=all clean.ivy.lo
</tasks>
```

[top](#)

## <exec>

Runs a specified command. The build fails if the command cannot be run or if it returns an error.

All paths specified are relative to the pipeline working directory.

## Attributes

Attribute	Required	Description
command	Yes	The command or script to be executed, relative to the working directory
args	No	Set of arguments (as a single string) to be passed to the command or script. Note that for complex or quoted arguments we suggest that you use separate <arg> tags for each argument.
workingdir	No	The directory in which the script or command is to be executed. Note that this directory is relative to the directory where the agent checks out the materials.

## Examples

- Invoke ruby, specifying the working directory as **tools/my-ruby-tool** and executing the ruby script **backup.rb**.

```
<tasks>
  <exec command="/usr/local/bin/ruby" args="backup.rb" workingdir="tools/my-ruby"
```

```
</tasks>
```

[top](#)

## **<arg>**

Specify a single argument for [exec](#) command.

This element is optional and can occur multiple times. It serves as an alternative to the "args" attribute of [exec](#), but it allows the use of any character required for making argument. For example, you can specify double quote using the xml escaped format: "

**Note:** When running commands on Windows, Go won't launch your command with system shell (cmd.exe), so you can't use shell commands (like echo) directly. If you want, you can pass your shell command as arguments to the cmd.exe.

On Windows you should specify the full name of your script file such as "mybuild.bat".  
(Only specifying "mybuild" won't work)

## **Examples**

- Echo something on Windows:

```
<exec command="cmd">
  <arg>/c</arg>
  <arg>echo</arg>
  <arg>something to print out</arg>
</exec>
```

- Run command with pipe character in arguments:

```
<exec command="MsBuild">
  <arg>D:\projects\project\project-8.sln</arg>
  <arg>/REBUILD</arg>
  <arg>/CFG="Release_99|Win32"</arg>
</exec>
```

[top](#)

## <nant>

Specifies a NAnt build to run. NAnt is assumed to be present from the command line on the agent.

All paths specified must be relative to the pipeline working directory.

## Attributes

Attribute	Required	Description
buildfile	No	Path to NAnt build file. If not specified, the path defaults to 'default.build'. The path is relative to the sandbox directory and cannot be outside the sandbox.
target	No	NAnt target(s) to run. If not specified, defaults to the default target of the build file.
workingdir	No	The directory from where NAnt is invoked
nantpath	No	Path of the directory in which NAnt is installed. By default Go will assume that NAnt is in the system environment variable \${PATH}. If the path is specified, then the path must be the same in all agents which run the job.

## Examples

Invoke NAnt, specifying a set of targets to run:

```
<tasks>
  <nant buildfile="myproject.build" target="smoke-test"/>
</tasks>
```

[top](#)

## <rake>

Specifies a Rake build to be run. Ruby and Rake are assumed to be present from the command line on the agent.

All paths specified must be relative to the pipeline working directory.

## Attributes

Attribute	Required	Description
buildfile	No	Path to Rake file. If not specified, the path defaults to 'rakefile'. The path cannot start from ''
target	No	Rake target(s) to run. If not specified, defaults to the default target of the build file
workingdir	No	The directory from where Rake is invoked

## Examples

Invoke rake, specifying a set of targets to run:

```
<tasks>
  <rake buildfile="rakefile" target="smoke-test"/>
</tasks>
```

[top](#)

## <fetchartifact>

Fetch artifacts from:

- 1. previous stages in the same pipeline, or
- 2. stages of pipelines that this pipeline depends on, directly or indirectly (ancestor pipelines).

When pointed to parent/ancestor pipeline, fetch task can pull artifacts from the upstream-stage or stages before it. This restriction has been introduced in 12.2. Stages after the upstream stage can not be fetched from, because they may not be complete when the fetch call executes.

All file paths specified are relative to the pipeline working directory.

## Attributes

Attribute	Required	Description
		This value can either be: 1. the name of upstream pipeline on which the pipeline of the job depends

pipeline	No	on. The pipeline should be added as a dependency under <materials> , or 2. the hierarchy of an ancestor pipeline of the current pipeline. Example, The value "BuildPipeline/AcceptancePipeline" denotes that the fetch task attempts to fetch artifacts from its ancestor 'BuildPipeline'. The given hierarchy denotes that the current pipeline depends on 'AcceptancePipeline' which in turn depends on 'BuildPipeline' using the dependency material definition given under materials. Defaults to current pipeline if not specified.
stage	Yes	The name of the stage to fetch artifacts from
job	Yes	The name of the job to fetch artifacts from
srcdir	One of srcdir/srcfile	The path of the artifact directory of a specific job, relative to the sandbox directory. If the directory does not exist, the job is failed
srcfile	One of srcdir/srcfile	The path of the artifact file of a specific job.

Note: If the file does not exist, the job will fail. Go will not fetch the artifact again if it has not changed. The directory path is relative to the pipeline working directory. I I dest I No I The path of the directory where the artifact is fetched to. The directory is overwritten if it already exists. The directory path is relative to the pipeline working directory. I

## Example:

1. Fetch all artifacts in the directory 'pkg' from the previous stage in the same pipeline and put them under the directory 'lib'

```

<pipelines>
  <pipeline name="go">
    ...
    <stage name="dev">
      <jobs>
        <job name="unit">
          <artifacts>
            <artifact src="target/deployable.jar" dest="pkg"/>
          </artifacts>
        </job>
      </jobs>
    </stage>
    <stage name="ft">
      <jobs>
        <job name="functional">
          <tasks>
            <fetchartifact stage="dev" job="unit" srcdir="pkg" dest="lib"/>
          </tasks>
        </job>
      </jobs>
    </stage>
  </pipeline>
</pipelines>

```

```
</pipeline>
</pipelines>
```

2. Fetch a single artifact from a stage in the upstream pipeline 'framework' and put it under the directory 'lib'

```
<pipeline name="go">
  <materials>
    <pipeline name="framework" stage="ft"/>
  </materials>
  <stage name="dev">
    <jobs>
      <job name="unit">
        <tasks>
          <fetchartifact pipeline="framework" stage="dev" job="unit"
            srcfile="pkg/deployable.jar" dest="lib" />
        </tasks>
      </job>
    </jobs>
  </stage>
</pipeline>
```

3. Fetch a single artifact from a stage in an ancestor pipeline 'Build' and put it under the directory 'pkg'

```
<pipeline name="deploy">
  <materials>
    <pipeline name="acceptance" stage="ft"/>
  </materials>
  <stage name="deploy-pkg">
    <jobs>
      <job name="deploy-win">
        <tasks>
          <fetchartifact pipeline="build" stage="dist" job="create-installer"
            srcfile="installers/deployable-setup.exe" dest="installer" />
        </tasks>
      </job>
    </jobs>
  </stage>
</pipeline>
```

[top](#)

## <runif>

Specifies when a task should be allowed to run. Multiple conditions may be defined for

each task.

A running job on an agent has two possible states: passed or failed. A job starts in the state “passed”. If any task fails, it transitions to the state “failed”.

A task can specify any of three possible runif filters: 'passed', 'failed' or 'any'. ('passed' is the default)

## Attributes

Attribute	Required	Description
status	Yes	The status is the job's current status. The value should be one of 'passed', 'failed' or 'any'.

## Notes:

`<runif>` can also be defined under `<exec>` task even if `<exec>` has some `<arg>`s e.g.

```
<exec command="echo">
  <arg value="test" />
  <runif status="passed" />
</exec>
```

## Examples

Given the tasks in a job is following:

```
<tasks>
  <ant buildfile="build.xml" target="complie-test-source"/>
  <fetchartifact pipeline="my_app" stage="dist" job="package-artifact" srcdir="pkgs">
    <runif status="passed"/>
  </fetchartifact>
  <exec command="../copy_error_log_to_somewhere" >
    <runif status="failed"/>
  </exec>
</tasks>
```

## Scenario one:

If task 1 `<ant>` passed, task 2 `<fetchartifact>` would be executed.

If task 2 `<fetchartifact>` passed, task 3 `<exec>` would NOT be executed.

If task 2 `<fetchartifact>` failed, task 3 `<exec>` would be executed.

## Scenario two:

If task 1 `<ant>` failed, task 2 `<fetchartifact>` would NOT be executed.

Instead, task 3 `<exec>` would be executed.

[top](#)

## `<oncancel>`

Specifies a task to execute when a stage is cancelled. Only one task can be defined in `<oncancel>`.

If a job is cancelled during the assigning phase, the job will not start preparing

If a job is cancelled during the preparing phase, preparing will complete, but no tasks will be executed

If a job is cancelled during the building phase:

- If the currently running task **does not** have `<oncancel>` defined, the task will be killed
- If the currently running task **does** have `<oncancel>` defined, the task defined within `<oncancel>` will execute immediately. As soon as both the original task and the `<oncancel>` task are completed, no other tasks will execute

If a job is cancelled during the completing phase, the agent will ignore the request and complete as planned

## Examples

The task 'start\_server' starts a process on an agent. When the stage is cancelled, the agent will invoke the cancel task 'kill\_server' to kill the process early and clean up any extra files.

```
<tasks>
  <ant target="start_server">
    <oncancel>
```

```

<ant target="kill_server" />
</oncancel>
</ant>
</tasks>

```

[top](#)

## <artifacts>

---

<artifacts> specifies what files the agent will publish to the server.

## Examples

```

<job name="unit">
  <artifacts>
    <artifact src="target/deployable.jar" dest="pkg"/>
    <test src="target/junit-output" dest="junit"/>
  </artifacts>
</job>

```

[top](#)

## <artifact>

---

Publish build artifacts to the artifact repository for the job.

## Attributes

Attribute	Required	Description
src	Yes	The file or folders to publish to the server. Go will only upload files that are in the working directory of the job. You can use wildcards to specify the files and folders to upload: * means any path, means any file or folder name.
dest	No	The destination is relative to the artifacts folder of the current instance on the server side. If it is not specified, the artifact will be stored in the root of the artifacts directory.

You can use wildcards to specify which files to upload. The wildcard syntax follows the commonly used ant/nant style. So "target/\*\*/\*.xml" would upload all xml files in the target directory and any of its subdirectories. The original directory structure is preserved on

the server.

## Examples

```
<job name="unit">
  <artifacts>
    <artifact src="target/deployable.jar" dest="pkg"/>
  </artifacts>
</job>
```

```
<job name="unit">
  <artifacts>
    <artifact src="target/**/*Test.xml" dest="pkg"/>
  </artifacts>
</job>
```

The following will upload all xml files to the server's artifact repository.

```
<job name="unit">
  <artifacts>
    <artifact src="target/**/*.xml" />
  </artifacts>
</job>
```

[top](#)

### <test>

The src attribute should point towards a folder that contains the test output files. Go will use these to generate a test report. Test information is placed in the Failures and Test sub-tabs. Test results from multiple jobs are aggregated on the stage detail pages. This allows you to see the results of tests from both functional and unit tests even if they are run in different jobs.

## Attributes

Attribute	Required	Description
src	Yes	Specify the directory into which the test output of the job will be put on agent side. This is relative to the Job's working directory.

dest	No	The path in the artifacts repository where the reports will be placed.
------	----	--

## Examples

```
<job name="unit">
  <artifacts>
    <test src="target/junit-output" dest="junit"/>
  </artifacts>
</job>
```

[top](#)

## <tabs>

The `<tabs>` element allows you to add tabs to the Job Details page. You can put any artifact that can be rendered by a web browser into a tab. For example, if your coverage tool produces an html report, you can easily place that report into a tab. Tabs are implemented as iframes (see W3C iframe definition ).

## Example:

```
<job name="unit">
  <artifacts>
    <artifact src="target/jcoverage" dest="Jcoverage"/>
  </artifacts>
  <tabs>
    <tab name="coverage" path="Jcoverage/index.html"/>
  </tabs>
</job>
```

[top](#)

## <tab>

Define a tab with specific name and artifact to show.

## Attributes

Attribute	Required	Description

name	Yes	The name of the tab. If should be unique in that job.
path	Yes	The relative path of a file in the artifact repository of the job.

## Example:

Given some coverage information in 'target/Jcoverage' folder on the agent side, We configure a tab to show the coverage information by specifying a tab with the index.html file.

```
<job name="unit">
  <artifacts>
    <artifact src="target/jcoverage" dest="Jcoverage"/>
  </artifacts>
  <tabs>
    <tab name="coverage" path="Jcoverage/index.html"/>
  </tabs>
</job>
```

## <properties>

The `<properties>` element allows you to create properties of the build from XML files or artifacts created during your build. You can export the values of properties over time. This allows you to track properties against certain builds, for example to see whether build time is improving or getting worse.

## Example:

```
<job name="emma">
  <artifacts>
    <artifact src="target/emma" dest="analysis" />
  </artifacts>
  <tasks>
    <ant target="emma">
  </tasks>
  <properties>
    <property name="coverage.class" src="target/emma/coverage.xml" xpath="substring-1"/>
    <property name="coverage.method" src="target/emma/coverage.xml" xpath="substring-1"/>
    <property name="coverage.block" src="target/emma/coverage.xml" xpath="substring-1"/>
    <property name="coverage.line" src="target/emma/coverage.xml" xpath="substring-be">
  </properties>
</job>
```

## <property>

Define a Property based on the contents of an XML file.

### Attributes

Attribute	Required	Description
name	Yes	The name of the property. It has to be unique within a <job>. The name can contain: a-z, A-Z, 0-9, fullstop, underscore and hyphen only. Spaces are not allowed. Name is case-sensitive.
src	Yes	The xml file containing the data that you want to use to create the property, and it isn't allowed to start from ''

Properties are set on the Agent at the end of the build and does not need to be an artifact that will be uploaded to the server. | | xpath | Yes | The XPath that will be used to create the property. |

### Example:

This is a simple example to parse the errors and failures count from a single junit file and turn them into properties.

```
<job name="junit">
  <tasks>
    <ant target="unittest">
  </tasks>
  <properties>
    <property name="junit.errors" src="target/junit-reports/TEST-MainAppTest.xml" xpath=".//error"/>
    <property name="junit.failures" src="target/junit-reports/TEST-MainAppTest.xml" xpath=".//failure"/>
  </properties>
</job>
```

Here's a more complex example. This will parse the class, method, block, and line coverage out of an [EMMA](#) coverage.xml file.

```
<job name="emma">
  <artifacts>
```

```

<artifact src="target/emma" dest="analysis" />
</artifacts>
<tasks>
  <ant target="emma">
</tasks>
<properties>
  <property name="coverage.class" src="target/emma/coverage.xml" xpath="substring-1"/>
  <property name="coverage.method" src="target/emma/coverage.xml" xpath="substring-2"/>
  <property name="coverage.block" src="target/emma/coverage.xml" xpath="substring-3"/>
  <property name="coverage.line" src="target/emma/coverage.xml" xpath="substring-bean"/>
</properties>
</job>

```

[top](#)

## <approval>

Specifies how a stage should be triggered. `<approval>` of type 'manual' or 'success' can be used to stop a pipeline execution at the start of a stage and can only be resumed when it is manually approved on the pipeline activity page, stage details page or through RESTful url.

## Attributes

Attribute	Required	Description
type	Yes	Either 'manual' or 'success'. 'manual' means the stage needs to be approved manually. 'success' means the stage will be automatically triggered when the previous stage passes.

### Notes:

- `<approval>` must be the first sub-element of `<stage>` .
- If an approval is not specified then the behavior is same as 'success' i.e. the stage will be automatically triggered when the previous stage passes.

## Example:

```

<stage name="ut">
  <approval type="manual" />
  ...
</stage>

```

[top](#)

## <authorization>

You can use `<authorization>` under an `<approval>` with a 'manual' or 'success' type to specify who can approve this stage. There are two sub-elements: `<user>` and `<role>`.

## Examples

```
<approval type="manual">
  <authorization>
    <user>lqiao</user>
    <role>go_admins</role>
  </authorization>
</approval>
```

```
<approval type="success">
  <authorization>
    <user>lqiao</user>
    <role>go_admins</role>
  </authorization>
</approval>
```

[top](#)

## <templates>

The `<templates>` element specifies the set of templates known by the server.

[top](#)

## <pipeline>

Allows you to provide a template for pipeline definition

## Attributes

Attribute	Required	Description
name	Yes	Identifier for the pipeline template

# Examples

```
<cruise>
  ...
  <pipelines group="studios" >
    <pipeline name="yourproject" template="project-template" >
      <materials>
        <svn url="http://your-svn/" />
      </materials>
    </pipeline>
  </pipelines>
  <templates>
    <pipeline name="project-template">
      <authorization>
        <admins>
          <user>jez</user>
        </admins>
      </authorization>
      <stage name="ut">
        <jobs>
          <job name="linux">
            <resources>
              <resource>linux</resource>
            </resources>
            <tasks>
              <ant target="unit-test" />
            </tasks>
          </job>
        </jobs>
      </stage>
    </pipeline>
  </templates>
</cruise>
```

[top](#)

## <environments>

The `<environments>` element specifies the set of environments known by the server.

[top](#)

## <environment>

Allows you to group a set of agents together for exclusive use.

# Attributes

Attribute	Required	Description
name	Yes	Identifier for an environment

## Examples

```
<cruise>
  ...
  <pipelines group="studios" />
    <pipeline name="yourproject" labeltemplate="foo-1.0.${COUNT}" >
      ...
    </pipeline>
  </pipelines>
  <environments>
    <environment name="UAT">
      <environmentvariables>
        <variable name="FOO"><value>bar</value></variable>
      </environmentvariables>
      <agents>
        <physical uuid="94fcb7ad-8b97-4078-b5f6-3c7436d6a390"/>
      </agents>
      <pipelines>
        <pipeline name="yourproject"/>
      </pipelines>
    </environment>
  </environments>
  <agents>
    <agent hostname="agent01" ipaddress="192.168.0.1" uuid="94fcb7ad-8b97-4078-b5f6-1">
    </agents>
  </cruise>
```

[top](#)

## <agents>

The `<agents>` element inside the `<environment>` element specifies the set of agents that it references.

[top](#)

## <environmentvariables>

`<environmentvariables>` specifies the **variables** to pass to jobs and their tasks. You can specify these on a `<pipeline>`, `<stage>`, `<job>` or an `<environment>`. If the same environment variable is defined either on the agent where the job runs or on the

pipeline/stage/environment of the job, the precedence is in the order `<job>`, `<stage>`, `<pipeline>`, `<environment>` and the system environment variable. For example, variable "FOO" defined in a job overrides the variable defined in the job's stage.

[top](#)

## `<variable>`

A `<variable>` defines the variable name and property value that will be passed to a job. It will be set on the system environment when the job is run. The value can be include multiple lines or CDATA. Note that the behaviour is operating system dependent. Your operating system may not allow certain variable names and/or values.

## Attributes

Attribute	Required	Description
name	Yes	Identifier for an environment variable
secure	No	This attribute is applicable only at the pipeline level and when set to true, encrypts the environment variable value.

## Example:

```
<environmentvariables>
  <variable name="FOO"><value>bar</value></variable>
  <variable name="MULTIPLE_LINES"><value>multiplelines</value></variable>
  <variable name="COMPLEX"><value>!<![CDATA[This has very <complex> data]]</value><!
</environmentvariables>
```

[top](#)

## `<physical>`

References a physical agent to be associated with this environment.

## Attributes

Attribute	Required	Description

uuid	Yes	Identifier to an agent (must exist in the config file).
------	-----	---

## Examples

```
<environment name="UAT">
  <agents>
    <physical uuid="94fcb7ad-8b97-4078-b5f6-3c7436d6a390"/>
  </agents>
</environment>
```

[top](#)

## <pipelines>

The `<pipelines>` element inside the `<environment>` element specifies the set of pipelines that it references.

[top](#)

## <pipeline>

References a pipeline to be associated with this environment.

## Attributes

Attribute	Required	Description
name	Yes	Identifier to an pipeline (must exist in the config file).

## Examples

```
<environment name="UAT">
  <pipelines>
    <pipeline name="yourproject"/>
  </pipelines>
</environment>
```

[top](#)

## <agents>

---

The `<agents>` element specifies the set of agents known by the server.

### Notes:

Do not change it manually. You can manage these through the Agents tab.

[top](#)

## <agent>

---

An approved agent. Before it is approved, the agent is displayed on the top of the agent tab with a grey bar.

## Attributes

Attribute	Required	Description
hostname	Yes	Name of your agent. This defaults to the hostname of the agent when it is approved.
ipaddress	Yes	IP for the agent.
uuid	Yes	Identifier for the agent. It is created by Go automatically.
isDisabled	No	The values should be one of 'true' or 'false' (or 1 / 0). 'true' or '1' means that the agent is denied. Go doesn't assign jobs to a denied agent.

### Notes:

A local agent will be approved automatically.

[top](#)

## <resources>

---

`<resources>` describes the resources available on a particular agent.

### Note:

An agent without any resources will build any jobs that don't specify resources. Refer to the `<resources>` of `<job>`.

[top](#)

## **<resource>**

---

resources names can contain the following characters: a-z, A-Z, 0-9, fullstop, underscore and hyphen. Spaces are not allowed.

## Examples

```
<agents>
  <agent hostname="agent01" ipaddress="192.168.0.1" uuid="94fcb7ad-8b97-4078-b5f6-3c"
    <resources>
      <resource>java</resource>
      <resource>linux</resource>
    </resources>
  </agent>
</agents>
```

# Schema

---

- Configuration file schema : [cruise-config.xsd](#)

## **ADVANCED**

---

## **USAGE**

---

# Auto registration of remote agents

As a Go administrator, you can auto approve remote agents by using a shared key between the Go Agent and Go Server.

- Add an attribute named "agentAutoRegisterKey", for e.g., agentAutoRegisterKey="388b633a88de126531afa41eff9aa69e", in the server configuration fragment.

```
<?xml version="1.0" encoding="utf-8"?>
<cruise xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="cruise-config.xsd" schemaVersion="41">
  <server artifactsdir="artifacts" purgeStart="1.0" purgeUpto="1.5" jobTimeout="60" agentAutoRegisterKey="388b633a88de126531afa41eff9aa69e">
```

- On the remote Go Agent machine, create a file named "autoregister.properties" under the < agent\_installation\_directory >/config directory and add the following contents:

```
agent.auto.register.key=388b633a88de126531afa41eff9aa69e
agent.auto.register.resources=ant,java
agent.auto.register.environments=QA
```

- Now, bringing up the remote agent should automatically register with the Go Server without the administrator having to 'Enable' the newly added agent.

# Run a Job on a group of Agents

---

Sometimes there is a particular job that you may wish to run on all agents in an environment, or in all agents that match a certain resource tag. For example you may want to run a system update on all linux agents, or install a new piece of software.

Go will run the Job on all agents that match the environment and resources specified in the job.

If an agent is missing or lost contact, a job will be scheduled. When the agent comes back on line, the job will be re-scheduled.

Jobs are given a unique name based on the name of the job in the configuration file. So for example, if the job is called 'deploy' and you have two agents, you would see jobs like 'deploy-runOnAll-1' and 'deploy-runOnAll-2'.

## Configure through the Admin UI

To enable run on all agents for a job, navigate to the Job settings page in the job configuration

[Job Settings](#)[Tasks](#)[Artifacts](#)[Environment Variables](#)[Custom Tabs](#)

## Job Settings

Job Name\*

deploy

Resources



Comma separated list

Job Timeout

 Never Use default (Never) Cancel after  minute(s) of inactivity

Run Type

 Run one instance Run on all agents Run  instances

\* indicates a required field

[RESET](#)[SAVE](#)

## Configure through the Config XML

To specify that a job should run on all agents, add the attribute

```
runOnAllAgents="true"
```

to the job's definition (see configuration reference for [`<job>`](#) )

```
<job name="deploy" runOnAllAgents="true">  
...
```

</job>

## Run 'X' instances of a Job

---

If you want to run multiple instances of the same job configuration you do not have to maintain multiple copies of same job config. You can specify how many instances of job you need & Go will take care of spawning the required number of job instances during scheduling.

This feature is particularly useful for test parallelization. It enables Go users to integrate with other test parallelization tools like [TLB](#) etc. to achieve distributed test execution with minimal configuration.

Jobs are given a unique name based on the name of the job in the configuration file. Example, if the job is called 'test' and you have set runInstanceCount to 2, you would see jobs like 'test-runInstance-1' and 'test-runInstance-2'. Go provides index of job (GO\_JOB\_RUN\_INDEX) & total count of jobs (GO\_JOB\_RUN\_COUNT) as environment variables to each Job.

## Configure through the Admin UI

To run 'x' instances of a job, navigate to the Job settings page in the job configuration

[Job Settings](#)[Tasks](#)[Artifacts](#)[Environment Variables](#)[Custom Tabs](#)

## Job Settings

Job Name\*

test

Resources

 [?](#)

Comma separated list

Job Timeout [?](#)

- Never
- Use default (Never)
- Cancel after  minute(s) of inactivity

Run Type

- Run one instance
- Run on all agents [?](#)
- Run  instances [?](#)

\* indicates a required field

[RESET](#)[SAVE](#)

## Configure through the Config XML

To specify that 'x' instances of a job should run, add the attribute

```
runInstanceCount="5"
```

to the job's definition (see configuration reference for [`<job>`](#) )

```
<job name="test" runInstanceCount="5">  
...
```

</job>

## Also See...

- [Re-running job\(s\)](#)

# Install multiple agents on the same machine

---

In order to fully utilize your hardware, it is often useful to install multiple agents on a single machine.

Currently Go installers do not support this out of the box. The following sections describe how this can be done manually

## Windows

---

On Windows, multiple Go agents can be run in two ways - as Windows service or as a Windows command

### Running as Windows service

- [Install your first agent with the installer to the default location](#)
- Copy the installation folder ("C:\Program Files\Go Agent") to "C:\Program Files\Go Agent 2"
- Delete the file C:\Program Files\Go Agent 2\config\guid.txt
- Delete the file C:\Program Files\Go Agent 2\agent-bootstrapper.running
- Edit **wrapper-agent.conf** file to customise information related to **Go Agent 2** Just after the line #include .../conf/wrapper-licenses.conf, add
  - set.GO\_AGENT\_DIR=C:\Program Files\Go Agent 2
  - set.GO\_AGENT\_JAVA\_HOME=%GO\_AGENT\_DIR%\jre
- Run the following command

```
sc create GoAgent2 binPath= "\"C:\Program Files\Go Agent2\cruisewrapper.exe\" -s
```

- Start "GoAgent2" service

### Running as Windows command

- Follow the instructions to [run Go without installation on Windows](#)
- Do the same steps on a different folder to set up another agent

You should use a VNC application (such as [TightVNC](#)) to keep windows user logged in. If the user logs out or the computer restarts, the agents will shutdown.

## Mac OSX

---

- [Install your first agent with the installer](#)
- Run Terminal.app
- Make an empty folder called go-agent-2
- In this folder, run

```
java -jar "/Applications/Go Agent.app/agent-bootstrapper.jar" 127.0.0.1 &
```

## Linux (and other UNIX)

---

- [Install your first agent with the installer](#)
- Make an empty folder called /var/lib/go-agent-2
- In this folder, run

```
java -jar /usr/share/go-agent/agent-bootstrapper.jar 127.0.0.1 &
```

# Clean up after canceling a task

---

When you have jobs that take a long time to run, it is very useful to have the capability to cancel it when you already know it will fail.

By default, Go will **kill any currently running tasks**. There are two other alternatives to this behaviour

- Specify a task to clean up your environment. This could kill the processes and cleanup any existing state.
- Indicate to Go you do not want anything done. This will allow the task to finish executing so the agent does not get into an inconsistent state.

## Using web interface

---

To perform a custom cleanup through the web interface, edit the desired **task configuration** and check the **On Cancel Task** checkbox in **Advanced Options**

The screenshot shows the 'Advanced Options' configuration dialog. It includes fields for 'Build file' (with a dropdown menu showing 'Ant'), 'Target' (with a help icon), 'Working directory' (with a help icon), and a section for 'Run if conditions\*' with checkboxes for 'Passed' (checked), 'Failed' (unchecked), and 'Any' (unchecked).

Advanced Options	
<input checked="" type="checkbox"/> On Cancel Task	Ant
Build file	(?)
Target	(?)
Working directory	(?)
Run if conditions*	
<input checked="" type="checkbox"/> Passed	<input type="checkbox"/> Failed
<input type="checkbox"/> Any	

# Using XML configuration

---

## Example: Override task to perform custom cleanup

Usage: As a developer, I want to stop running my [Twist](#) tests and clean up the environment on each job when I cancel the stage.

- On the [Administration Tab](#), edit the jobs that should handle canceling correctly
- Ensure the following "task" block is in the job configuration
- Now, whenever you cancel the stage while the jobs are running the ant "twist" target, the target "kill\_twist" will execute

## Example: Override task to disable all cleanup

Usage: As a developer, I want to allow my database tests to not be halted when I cancel the stage.

- On the [Administration Tab](#), edit the jobs that should handle canceling correctly
- Ensure the following "task" block is in the job configuration
- Now, whenever you cancel the stage while the jobs are running the rake "db-test" target, the agent will finish the task before picking up new work

# Conditional task execution

---

At times there are certain steps you need to execute only when you know that the build has already failed. For example, when a test suite fails you might want to output additional environment information to the console output

## Using web interface

---

Check the appropriate **Run if conditions** when defining the **Task**

---

### Basic Settings

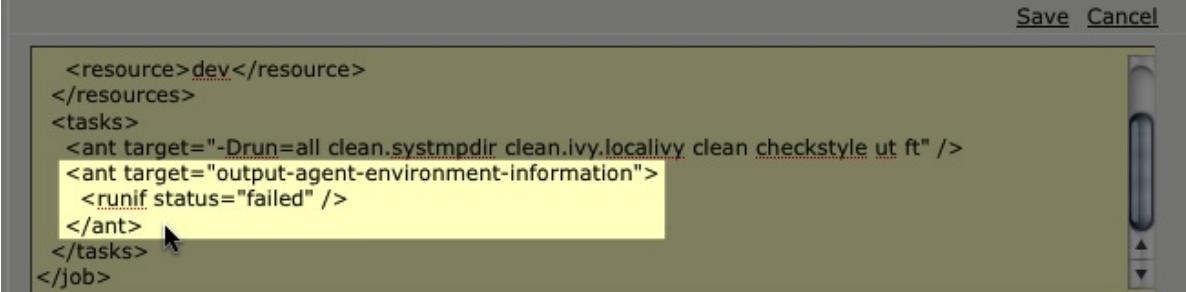
The screenshot shows a configuration form for a task. At the top, there are fields for 'Build file' (empty), 'Target' (set to 'output-agent-environment-information'), and 'Working directory' (empty). Below these is a section titled 'Run if conditions\*' which contains three checkboxes: 'Passed' (unchecked), 'Failed' (checked), and 'Any' (unchecked). The 'Failed' checkbox is highlighted with a red box.

## Using XML configuration

---

Usage: As a developer, I want to run a task only when the build has already failed.

- On the [Administration Tab](#), edit the jobs that should run a task when the build has failed
- Ensure the following "task" block is in the job configuration



The screenshot shows a graphical user interface for editing Jenkins job configurations. At the top right are 'Save' and 'Cancel' buttons. The main area contains XML code for a Jenkins job:

```
<resource>dev</resource>
</resources>
<tasks>
    <ant target="-Drun=all clean.systmpdir clean.ivy.localivy clean checkstyle ut ft" />
    <ant target="output-agent-environment-information">
        <runif status="failed" />
    </ant>
</tasks>
</job>
```

- Now we will get extra output only when the tests fail!

## Also See

---

- [Re-running job\(s\)](#)

# Trigger with a different revision of material

Go supports a Trigger with option that allows you to run the pipeline with a specific revision of the material(s).

## Trigger with options

The screenshot shows a 'website-build - Trigger' dialog box. In the 'Materials' tab, there is a section for 'BookWebsite\_GitSrc' with a revision history. The current revision is 4e24df6. A dropdown menu titled 'Revision to trigger with:' lists the last five revisions:

Revision	Committer	Comment	Date
4e24df6	kmugrage <kmugrage@thoughtworks.com>	added a new line to the README file	1 day ago
f04a3d0	kmugrage <kmugrage@thoughtworks.com>	Modified README file	1 day ago
713e8be	Nanda <freespirit@tw.com>	"Fixing the typo"	1 day ago
349bda4	Nanda <freespirit@tw.com>	"Made some changes on the UI"	4 days ago
64da752	Nanda <freespirit@tw.com>	"Fixing the typo"	6 days ago

At the bottom of the dialog, there are 'TRIGGER PIPELINE' and 'CLOSE' buttons.

## Information

The following information are displayed for the last 5 revisions. For an SCM material, the following information is shown

- Revision hash or pipeline label
- Committer
- Check-in comment for the revision
- Check-in time of the revision

For a pipeline material, the pipeline label and the corresponding run time is shown

## Choosing the revision

You can choose one of the revisions and then click on Trigger Pipeline button.

If you want to trigger with a revision other than the 5 that is displayed, you can specify information related to this in the text box provided. You can search for all or part of

- revision hash/pipeline label
- committer name
- check-in comment

Go will find matches and display the same. One of the matches can be chosen and the build triggered.

The screenshot shows the Jenkins 'website-build - Trigger' dialog. At the top, there's a 'Materials' tab. Below it, a table shows details for a commit: Git hash (4e24df6), Dest (not-set), Date (1 day ago), User (kmugrage <kmugrage@thoughtworks.com>), Comment (added a new line to the README file), and Last run with (4e24df6). A 'Revision to trigger with:' input field contains the text 'typo'. Below this, a list of three commits is shown:

Revision	Author	Date
713e8be	Nanda <freespirit@tw.com>	1 day ago "Fixing the typo"
64da752	Nanda <freespirit@tw.com>	6 days ago "Fixing the typo"
e9f5dd8	Nanda <freespirit@tw.com>	2 months ago "Fixing the typo"

At the bottom of the dialog are two buttons: 'TRIGGER PIPELINE' and 'CLOSE'.

## Environment and secure variables

If the pipeline has environment and/or secure variables configured, additional tabs will be displayed to allow you to override these values.

## website-build - Trigger

Materials

Environment Variables

Secure Variables

4e24df6

BookWebsite\_GitSrc

Git

BookWebsite\_GitSrc

Dest:

not-set

Date:

1 day ago

## Also see...

- [Deploy a specific build to an environment](#)
- [Pipeline dashboard](#)
- [Ordering of pipelines](#)

# Fan-in Dependency Management

---

Go supports fan-in dependency resolution for pipelines that are on auto trigger. Fan-in material resolution will ensure that a pipeline triggers only when all its upstream pipelines have triggered off the same version of an ancestor pipeline or material. This will be the case when you have multiple components building in separate pipelines which all have the same ancestor and you want downstream pipelines to all use the same version of the artifact from the ancestor pipeline.

When you have non trivial pipeline dependency graphs, significant differences in pipeline execution times and automatic builds/deployments, you may typically run into the following issues

- **Wasted builds** : Premature runs which do not have the right version of dependent components because one of the dependent pipelines was faster than the rest.
- **Inconsistent results** : Your deployment that depends on multiple components may have incompatible versions of components because the build times of these components are different
- **Incorrect feedback** : Your deployment to Production should happen only when it has successfully passed the UAT, Staging and Pre-Prod environments but it was triggered prematurely as soon as UAT went green.
- **Running code with the wrong tests** : Your commit to SCM contains both code and tests written for the code. Your pipelines are modeled such that your acceptance or test pipeline runs after the build pipeline. Acceptance has to run with the right tests for the code but instead it triggers as soon as the commit goes through with the previous available version for tests.

Go helps solve all of the above problems.

## How to use fan-in:

---

- In cases where your SCM material is used throughout the process you will need to define the same URL for the material throughout. This will let Go know that it is a shared material and Go will enforce fan-in wherever applicable. For example: code, tests, environment configuration are in <http://svn.company.com/code>, <http://svn.company.com/tests>, and <http://svn.company.com/config> respectively. In this case ensure for pipelines that need these materials, the url is set to the same value. For example the pipelines Build, Acceptance and Deploy have the material

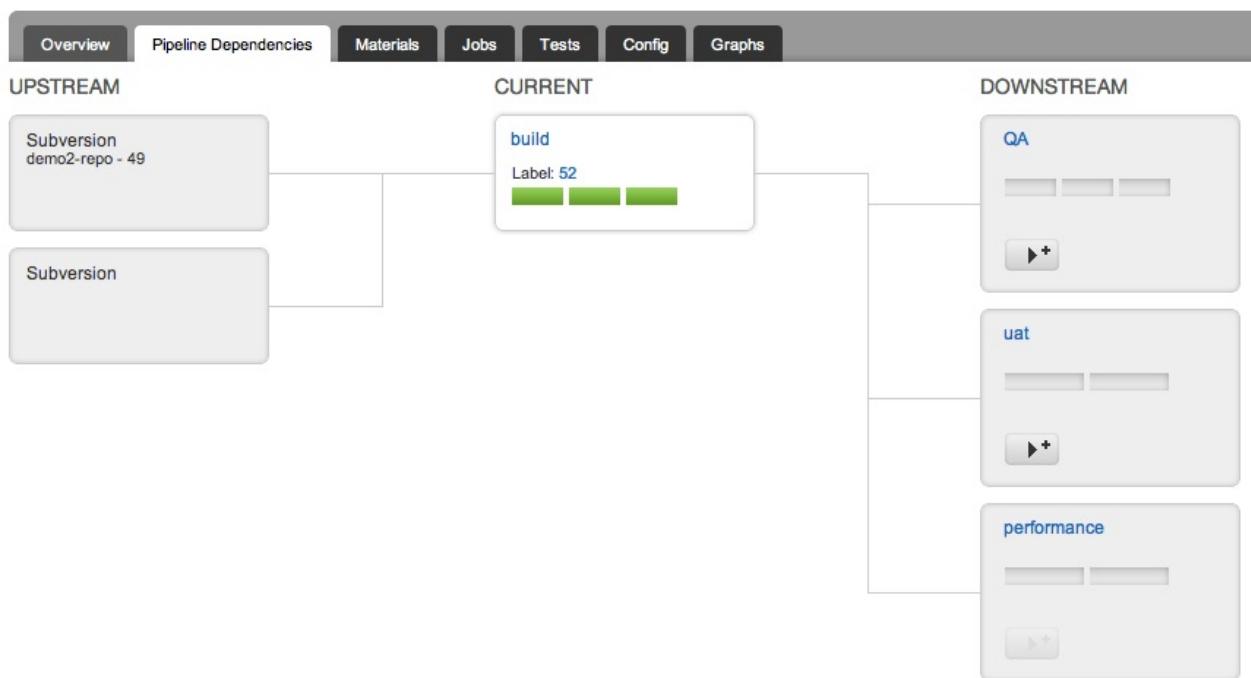
url <http://svn.company.com>

- Pipelines where fan-in dependency resolution is required will need to have trigger type set as auto

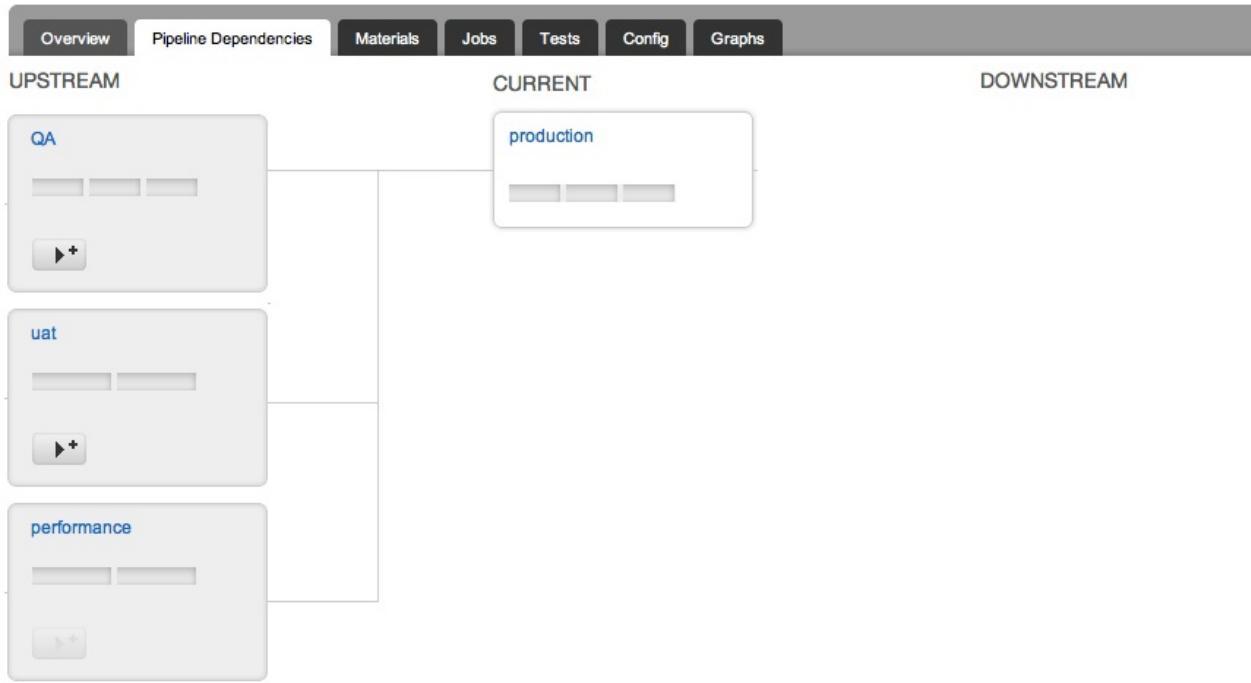
## Example use cases for fan-in resolution

### Creating a gate to production

I want to push a build into a number of environments in parallel (Manual QA, UAT for business sign off and performance testing) so that all these activities happen at the same time. So we have a pipeline dependency model as shown below



For the next step, when deploying to production, we have a pipeline that depends on all 3 of the above pipeline so that a build that has succeeded in all 3 environments is automatically pushed to production. This is shown below.



### Sequence of Events and Resolution

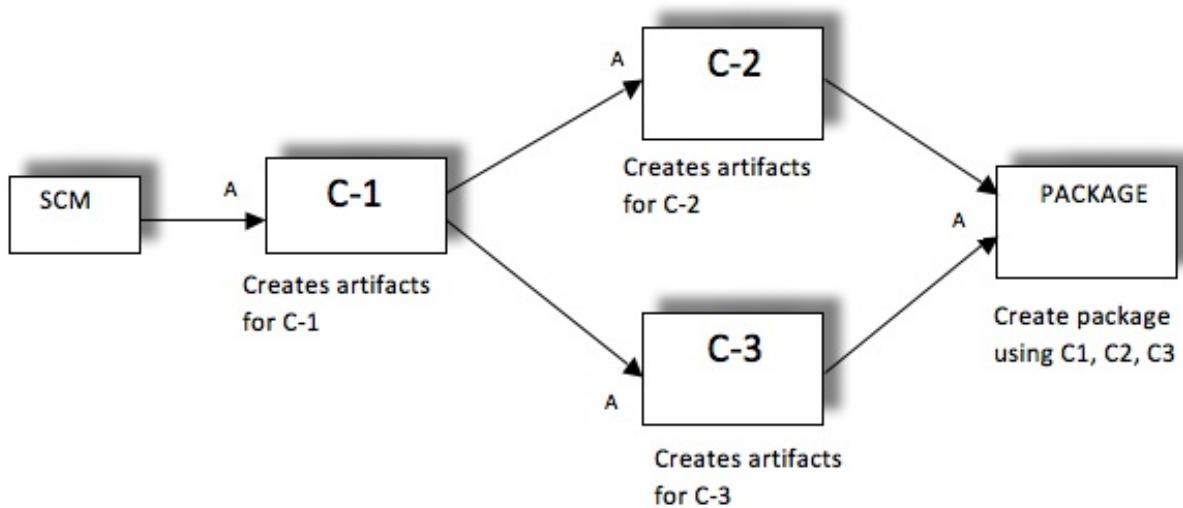
- A new build is generated.
- The build is pushed to QA, UAT and performance pipelines.
- Regardless of the time taken in each of these pipelines, Go will ensure that production is triggered only after all three pipelines go green with the same version of build. Production will use the right version of your build artifacts.

How to configure:

- Production should have trigger type as auto.
- QA, UAT and Performance should have Build as a material. Trigger type for these pipelines can be either manual or auto.

## Picking the right version of dependent components

I have three component pipelines (C1, C2, C3) and a package pipeline that fetches their artifacts and creates a deploy package. Components C2 and C3 depend on pipeline C1 and have it as a material. The pipeline for C2 builds quickly but C3 takes a while. So we have a pipeline dependency model as shown below



The package pipeline should not trigger as soon as C2 is done. It should trigger only if both C2 and C3 go green. Additionally Package should use the same version of C1 that was used by C2 and C3.

#### Sequence of Events and Resolution

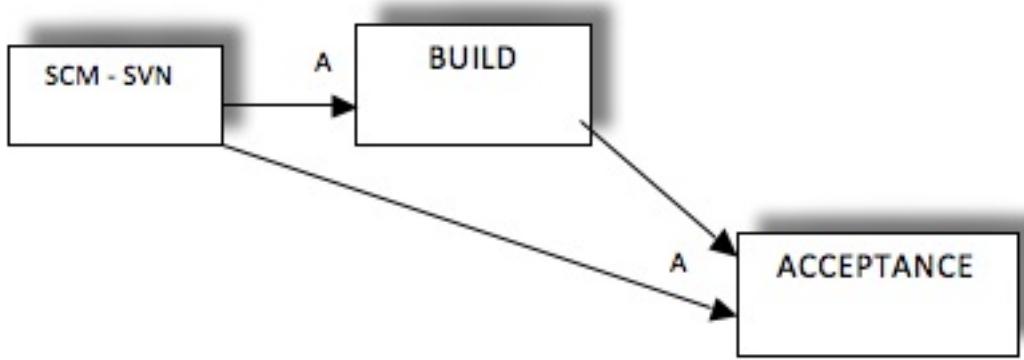
- C1 is triggered on a change.
- C2 and C3 are triggered after C1 is successful.
- C2 builds quickly but C3 is still in Progress.
- Go resolves that C3 and C2 depend on a common ancestor C1. Hence Go will wait for C3 to finish.
- If C3 goes green, the Package pipeline will trigger. It will use the right versions of C1, C2 and C3.

#### How to configure:

- Add C1 as a material for pipelines C2 and C3
- Add C2 and C3 as materials for pipeline Package.
- Package should have trigger type as auto

## **Test source code with the right version of tests**

You check-in code and tests as part of the same commit. The build pipeline compiles code and creates an artifact. The Acceptance pipeline fetches the build artifact and runs the tests that were written for the compiled code. Acceptance has to use the same tests committed with the code. So we have a pipeline dependency model as shown below



### Sequence of Events and Resolution

- On committing the changes, the Build pipeline will trigger with the latest revision.
- Although Acceptance also has the same material dependency, Go will not trigger it immediately.
- Build pipeline executes successfully.
- Acceptance will now trigger with the same version of the SCM and fetch the right build artifact from the Build pipeline.

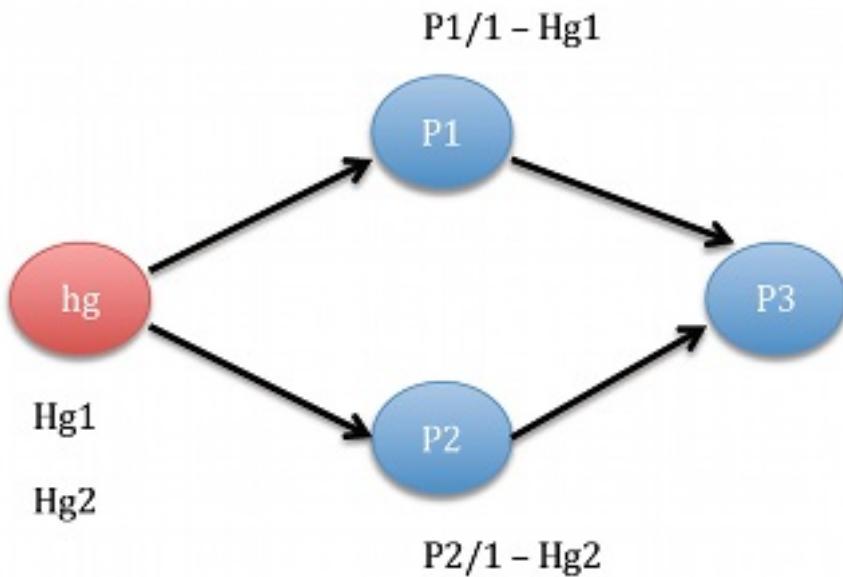
### How to configure:

- Add the same SCM material to pipelines Build and Acceptance i.e the same URL.
- Pipelines Build and Acceptance have trigger type as auto.

## Limitations

---

- **Fan-in and blacklist** : If the pipelines contributing to the fan-in have blacklist in their material definitions, the target pipeline does not adhere to fan-in behavior.



For example, refer to the scenario above. P1 and P2 are two pipelines which use the same mercurial (hg) repository, with two folders say "p1" and "p2". P1 is configured with "p2" folder in the blacklist. Likewise P2 is configured with "p1" folder in the blacklist.

The first run of P1 happens when a check-in happens in "p1" folder. The first run of P2 happens when there is a check-in to "p2".

In this scenario, P3, which is dependant on P1 and P2 does not trigger.

## Notes

- Fan-in as a feature is enabled by default. In case you need pipelines to trigger with every version regardless of ancestor versions you can disable fan-in. To disable fan-in you will need to add a system property and restart the Go server. On linux add the following line to /etc/default/go-server

```
export GO_SERVER_SYSTEM_PROPERTIES=''-Dresolve.fanin.revisions=N'
```

On windows, in the config folder of the Go server installation, edit the wrapper-server.conf file, and add an additional property with the value '-Dresolve.fanin.revisions=N'. For example:

```
wrapper.java.additional.17=''-Dresolve.fanin.revisions=N'
```

- Go will apply fan-in dependency resolution for pipelines that have auto trigger type only.

# Properties

## Introduction

Properties provide a simple way of collecting metrics over time. Go sets some standard properties for you. You can also set properties yourself using the Go REST APIs (see [Properties API](#) for more information). Go also allows you to download the history of your job in a convenient CSV format, so that you can analyse the results in spreadsheets or scripts.



The screenshot shows the Go interface with the 'Properties' tab selected. Below the tabs, there is a link to export the property history to a spreadsheet (CSV) and a link to this specific tab.

Property name	Property value
cruise_agent	b1stdcrspbs01.thoughtworks.com
cruise_job_duration	0
cruise_job_id	8310
cruise_job_result	Passed
cruise_pipeline_counter	21
cruise_pipeline_label	52
cruise_stage_counter	1
cruise_timestamp_01_scheduled	2010-06-28T15:13:22+05:30
cruise_timestamp_02_assigned	2010-06-28T15:14:02+05:30
cruise_timestamp_03_preparing	2010-06-28T15:14:12+05:30
cruise_timestamp_04_building	2010-06-28T15:14:17+05:30
cruise_timestamp_05_completing	2010-06-28T15:14:17+05:30
cruise_timestamp_06_completed	2010-06-28T15:14:17+05:30

## Property history

Go allows you to download the history of properties that you have defined. This history is available as a Comma Separated Values (CSV) file. You can import this file into a spreadsheet program to generate charts and diagnostics of your project.

You can of course access these resources through standard URLs:

- **CSV --**

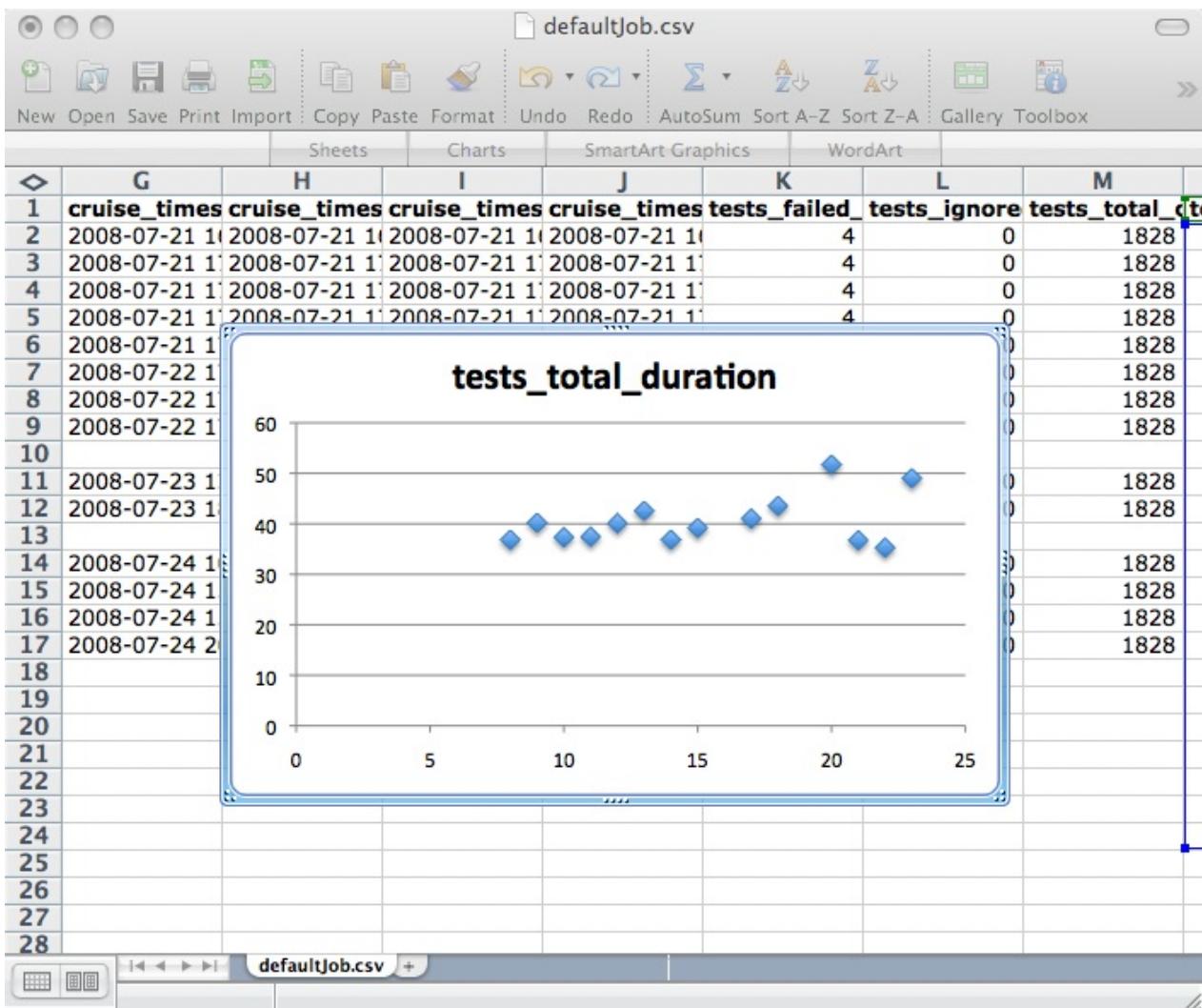
[http://\[server\]/go/properties/\[pipelineName\]/\[pipelineLabel\]/\[stageName\]/\[stageCount\]](http://[server]/go/properties/[pipelineName]/[pipelineLabel]/[stageName]/[stageCount])

er]/[job]/[propertyName]

To open the property history in a spreadsheet application, you can click on the **Export property history to spreadsheet (csv)** link on the Properties tab of the job.

The screenshot shows a spreadsheet application window titled "defaultJob.csv". The top menu bar includes "File" (New, Open, Save, Print, Import), "Edit" (Copy, Paste, Format), "Tools" (Undo, Redo, AutoSum, Sort A-Z, Sort Z-A), and "Format" (Sheets, Charts, SmartArt, Graphics, Word). The left sidebar lists properties: "cruise\_agent", "cruise\_job\_duration", "cruise\_job\_id", "cruise\_job\_result", "cruise\_pipeline\_counter", "cruise\_pipeline\_label", "cruise\_stage\_counter", "cruise\_timestamp\_01\_scheduled", "cruise\_timestamp\_02\_assigned", "cruise\_timestamp\_03\_preparing", "cruise\_timestamp\_04\_building", "cruise\_timestamp\_05\_completing", and "cruise\_timestamp\_06\_completed". The main table has columns A, B, C, and D. Column A contains row numbers from 1 to 19. Column B contains values like "cruise\_agent", "some-agent", and "cruise\_job". Column C contains status codes like "Failed", "Passed", and "0". Column D contains dates like "2008-07-15 19" and "2008-07-23 12".

	A	B	C	D
1	cruise_agent	cruise_job	cruise_job	cruise_pip
2	some-agent		6 Failed	1 2008-07-15 19
3	some-agent		37 Passed	2 2008-07-15 19
4	some-agent		29 Passed	3 2008-07-15 20
5	some-agent		33 Passed	4 2008-07-21 16
6	some-agent		28 Passed	5 2008-07-21 16
7	some-agent		27 Passed	6 2008-07-21 16
8	some-agent		31 Passed	7 2008-07-21 16
9	some-agent		30 Passed	8 2008-07-21 16
10	some-agent		32 Passed	9 2008-07-21 17
11	some-agent		30 Passed	10 2008-07-21 17
12	some-agent		32 Passed	11 2008-07-21 17
13	some-agent		39 Passed	12 2008-07-21 17
14	some-agent		39 Passed	13 2008-07-22 17
15	some-agent		32 Passed	14 2008-07-22 17
16	some-agent		34 Passed	15 2008-07-22 17
17	some-agent		0 Failed	16 2008-07-23 08
18	some-agent		44 Passed	17 2008-07-23 12
19	some-agent		44 Passed	18 2008-07-23 18



## Standard Properties

The standard properties defined by Go are:

- **cruise\_agent** -- the agent that is running the job
- **cruise\_job\_duration** -- total time to run the job
- **cruise\_job\_result** -- one of "passed" or "failed"
- **cruise\_job\_id** -- the name of the folder that the artifacts of the job was stored in under the artifact repository on server side (on earlier versions of Go).
- **cruise\_pipeline\_label** -- same as the value of the environment variable GO\_PIPELINE\_LABEL
- **cruise\_pipeline\_counter** -- same as the value of the environment variable GO\_PIPELINE\_COUNTER
- **cruise\_stage\_counter** -- same as the value of the environment variable GO\_STAGE\_COUNTER
- **cruise\_timestamp\_01\_scheduled** -- time at which the job was scheduled
- **cruise\_timestamp\_02\_assigned** -- time at which the job was assigned to the agent

- **cruise\_timestamp\_03\_preparing** -- time at which the job entered the "preparing" state
- **cruise\_timestamp\_04\_building** -- time at which the job started building
- **cruise\_timestamp\_05\_completing** -- time at which the job entered the completing state
- **cruise\_timestamp\_06\_completed** -- time at which the job completed

## Generating Properties from Artifacts

---

Go allows you to generate properties from XML artifacts that you create during the build. This can be used to harvest statistics produced by coverage tools etc. By storing them as properties it becomes very easy to show the history and trends over time of these values.

Note that the properties are generated on the agent side, so the src path is relative to the working directory of the pipeline on the agent.

For example, to add support for the coverage tool "Emma", you might do this:

```
<job>
  <properties>
    <property name="coverage.class" src="target/emma/coverage.xml" xpath="substring-1"/>
    <property name="coverage.method" src="target/emma/coverage.xml" xpath="substring-1"/>
    <property name="coverage.block" src="target/emma/coverage.xml" xpath="substring-1"/>
    <property name="coverage.line" src="target/emma/coverage.xml" xpath="substring-be-1"/>
  </properties>
</job>
```

## Tests

If you define a tests artifact that contains the test reports, then Go will add some properties associated with the tests.

- **tests\_failed\_count** -- number of failed tests
- **tests\_ignored\_count** -- number of ignored tests
- **tests\_total\_duration** -- total time taken for the tests
- **tests\_total\_count** -- total number of tests

tests_failed_count	4
tests_ignored_count	0
tests_total_count	1828
tests_total_duration	49.034

# Compare Builds

---

Go allows you to compare any two builds of a pipeline and see exactly what changes happened between those two instances. The information in this view will include:

- Code checkins
- Upstream pipelines
- Story/defect numbers (when linked to a tracking tool)

## Accessing Pipeline Compare

---

There are several locations from where the Pipeline Compare feature can be invoked. These include:

- Pipelines Dashboard page (the "Compare" link in each pipeline)
- Environments page
- Stage Details page
- Stage History widget within the Stage Details page

## Understanding the Pipeline Compare Screen

---

The Pipeline Compare screen lets you compare any two instances of a pipeline. Every pipeline instance is associated with a set of changes; be it a source control modification or an upstream pipeline. Performing a compare lets you easily identify exactly what these changes were.

Automatically triggered on 29 Mar, 2011 at 17:04:00 [+0530]

Changes Card Activity

Pipeline - publish-rpm

Revision	Label	Completed at
publish-rpm/41/publish/1	41	2011-04-06T17:19:18+05:30

Mercurial - https://cruise\_builder:\*\*\*\*\*@birstdscm01.thoughtworks.com/hg/cruise

Revision	Modified by	Comment
1b29e6ea1453a30 941f95710bfeeb4c8bffd530	PS 2011-04-05T11:25:01+05:30	#4839 - Added a test to document the behaviour that we do not show a material if it has not changed since the 'from' counter. This caused some confusion but we decided that this is how the diff behaviour should be i.e. do not show a material if it has not changed. We can potentially explore the option of showing the material but saying its not changed. May be later.
43d1f3122ef03f6495f26cc cd38d8919cea640	jemarley 2011-04-05T05:10:34+05:30	added more links for Go->Mingle help Iuuu/#308
9fdfe78667c149b3ef36d0d 551523136b639890	PS 2011-04-04T13:58:15+05:30	#4914 - add has_tree_view class to admin_workspace if you want to add the margin. Also change the tree selection such that the parent nodes are styled with a parent_selected class.

- To/from search box:** You can search for the appropriate pipeline instance using any of the following - pipeline label, check-in comment, person who checked in, upstream pipeline label and revision.
- Upstream dependency changes:** All the changes to upstream pipelines within the search range.
- Changes to version control systems:** All check-ins that went into dependent VCS materials within the search range.
- Tracking tool integration:** If you've configured a [tracking tool integration](#) for this pipeline or any upstream pipelines, check-in comments containing story/defect/ticket numbers would be hyperlinked to the appropriate tracking tool.

## Using Pipeline History

You can also select a pipeline by browsing the pipeline history.

Select a pipeline to compare

X

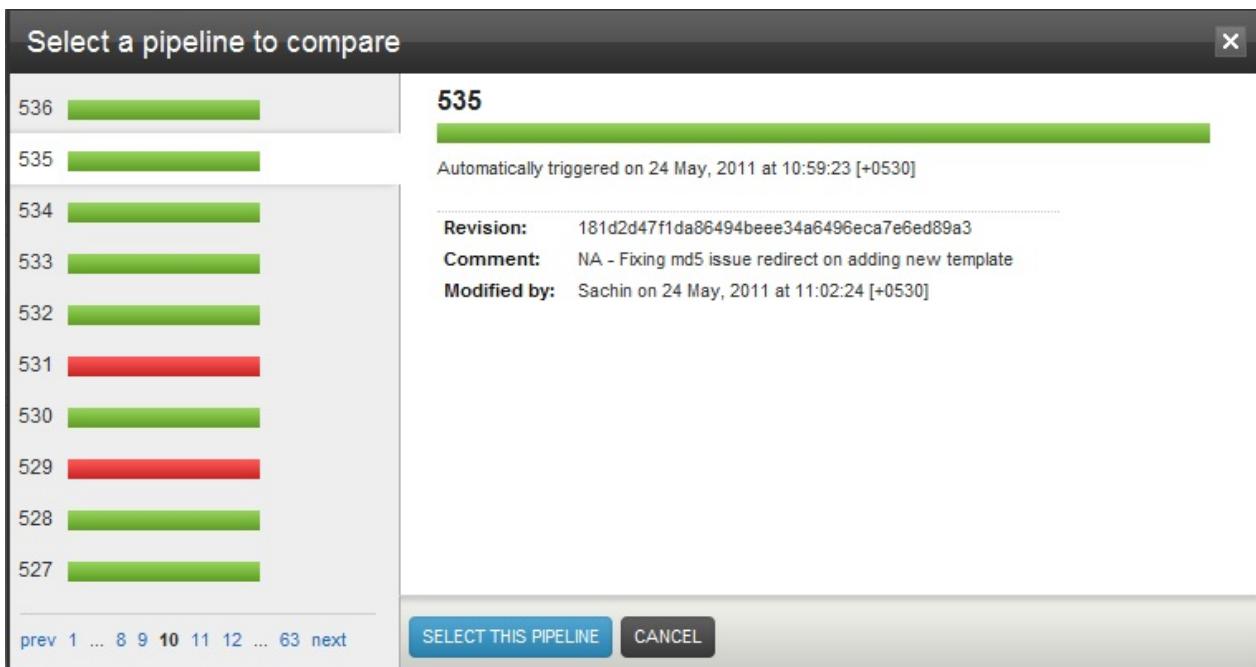
Pipeline ID	Revision
536	
535	
534	
533	
532	
531	
530	
529	
528	
527	

Automatically triggered on 24 May, 2011 at 10:59:23 [+0530]

**Revision:** 181d2d47f1da86494beee34a6496eca7e6ed89a3  
**Comment:** NA - Fixing md5 issue redirect on adding new template  
**Modified by:** Sachin on 24 May, 2011 at 11:02:24 [+0530]

prev 1 ... 8 9 10 11 12 ... 63 next

**SELECT THIS PIPELINE** CANCEL



Steps to select a particular instance from the history:

1. Click on the search box
2. Click on "Browse the timeline"
3. Browse the history and select a pipeline

## See Also

- [Mingle card activity gadget](#)

# Graphs

# Stage Duration Chart

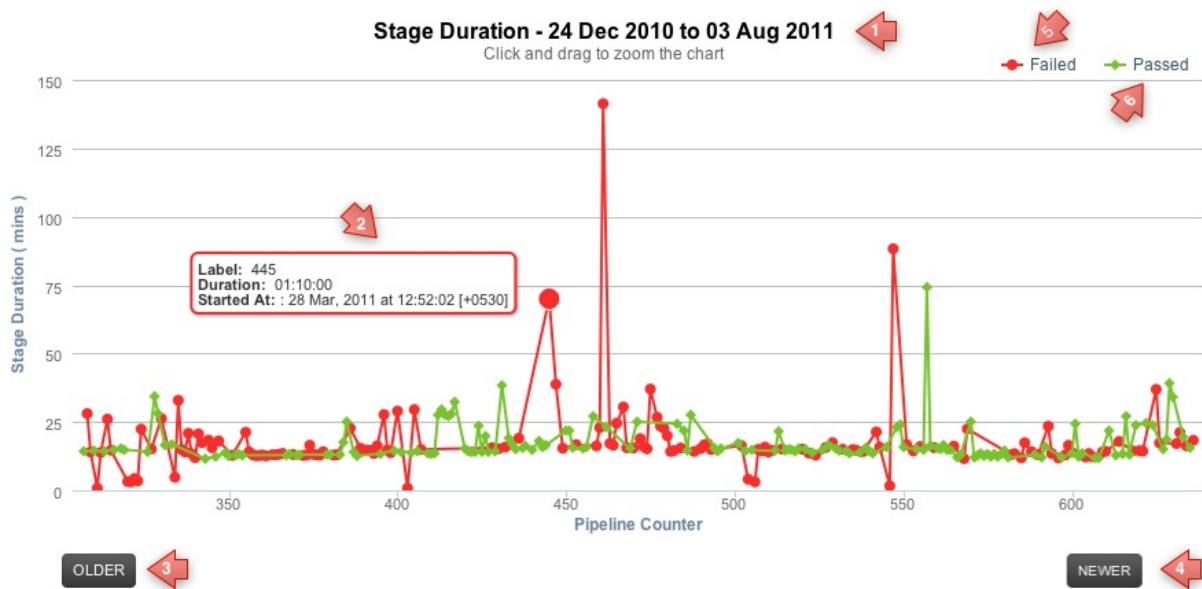
Go provides a chart on the stage details page which can be used for trend analysis of the stage's periodic runs. This graph shows two line graphs, one each for passed and failed stage instances which are plotted using the total duration of the last 300 stage instances.

## To navigate to the Graphs:

1. Click on the stage bar of the relevant stage on the pipelines dashboard.
  2. Click on the tab 'Graphs'.

## Features of the Graph

The following is a snapshot of a stage duration chart



Selecting a part of the graph zooms into the selected area. Once zoomed, a link called 'Reset Zoom' appears on the chart which resets the graph.

### **Key:**

1. The title shows the start and end dates of the range of the chart.

2. A tooltip giving the details of the stage run appears on hover over the points of the line graph.
3. Navigates to the Chart of older 300 runs.
4. Navigates to the Chart of newer 300 runs.
5. Toggles the line graph of the failed runs.
6. Toggles the line graph of the passed runs.

# Command Repository

## Introduction

This feature is an enhancement to [Custom Commands](#) in Go. Rather than start from scratch with command name and arguments, you now have the option to look up a command from a repository of useful commands maintained by the Go team. The lookup provides a starting point, you may need to edit the arguments, specify a working directory etc. Once saved, the behaviour is the same as a manually entered custom command. Take a look at this [video](#) to learn more.

## Using the command repository

This feature can be used anywhere you need to specify a custom command.

- Add a new pipeline

The screenshot shows the 'Add Pipeline' interface. On the left, there's a form for creating a new job. It includes fields for 'Job Name\*' (set to 'defaultJob'), 'Task Type\*' (set to 'More...'), 'Command\*', and 'Arguments'. Below the arguments field, there's a note: 'Enter each argument on a new line'. There's also a 'Working Directory' field. On the right, there's a 'Lookup Commands' sidebar containing a list of commands: build, buildr, gradle, maven clean, maven compile, maven package, maven test, and msbuild windows. The 'build' command is currently selected.

- Add a new stage to a pipeline
- Add a new job to a stage

- Add a new task to a job
- Edit a task

### Custom commands and agents

Go does not check if the command that you have specified is available on the agents.

Install the relevant command on all the agents where you want it to run. If it is not available on the default path, add it to the path in /etc/default/go-agent or equivalent. This requires an agent restart.

Alternatively, create a symbolic link to the path to the executable from /bin or equivalent folder. This doesn't require an agent restart

## Args style commands

The older `args` style commands are not supported by this feature. Please convert them to the new syntax using the config xml editor ( **Admin > Config XML** ). For example:

```
<exec command="touch" args="a b c"/>
```

becomes

```
<exec command="touch">
  <arg>a</arg>
  <arg>b</arg>
  <arg>c</arg>
</exec>
```

## Bundled Repository

The default set of commands come from <https://github.com/gocd/go-command-repo>. This repository is maintained by the Go team. The Go server installer bundles a clone of this Git repository under `<server-install-root>/db/command_repository/default`. Every upgrade of Go Server will overwrite the contents of this directory with an up to date set of commands. Hence, please do not add your commands here. Instead, set up a [private repository](#).

# Pulling Updates

Go team will continue to add (and sometimes update) commands to the repository on GitHub. If you want to make these commands available to your Go server without waiting for a new release or without upgrading your Go server, you could git pull them into `<go-server-install-root>/db/command_repository/default` as desired.

## Linux/Unix

Here is a simple cron tab entry that you could add to Go service account's cron tab to pull commands once a day.

```
@daily cd < go-server-install-root >/db/command_repository/default;git pull >>/var/gc
```

**Caution:** Don't pull as root/administrator. Use the Go service account.

## Windows

On Windows, you could set up a scheduled task to run this script on a schedule.

```
echo %date% %time% >>c:\pull-log.txt 2>&1
cd "C:\Program Files (x86)\Go Server\db\command_repository\default"
git pull >>c:\pull-log.txt 2>&1
```

Go caches these commands with a refresh interval of 30 minutes so you may not see the results of a pull immediately in the lookup unless you hit the reload cache button under the command repository section on the server configuration admin page or by using the [reload API](#).

## Private Repository

If you want to add your own commands for look up, you should set up your own Git/Mercurial/Subversion/Perforce/TFS repository that your command authors can commit/check-in into.

Make a clone/working copy available under `<go-server-install-root>/db/command_repository/<your-repo-name>`. Symbolic links are not supported. Then switch Go Server to this location. To do this, go to **Command Repository Management**

section of **Server Configuration** in **Admin** tab and change the **default** value



From the Go server's point of view, the command repository is just a directory under which it recursively looks for valid command.xml files. Note that directory names starting with a dot will be ignored.

Go will not lookup from the bundled repository if you switch to your own repository. You could choose to manually seed your private command repository with Go's bundled set of commands if you want to have them in addition to your own commands.

## Recommended process

1. Command author pushes/checks-in command to corporate version control system
2. Cron job on Go-server pulls/updates local repository/working copy in due course
3. Go Server caches the commands to improve response time. In case of changes in the command repository, new command gets into the cache in one of the following ways:
  - i. The default cache invalidation interval of 30 mins kicks in and the cache gets refreshed
  - ii. Go server admin clicks on the **RELOAD CACHE** button
  - iii. Go server admin uses the **reload API** through a cron job or otherwise to force a cache reload.

The commands in the command repository are not part of your Go Server config. They become part of your Go server config only after you (optionally edit and) save them.

## Command syntax and lookup logic

The screenshot shows the Go Server interface. On the left, there is a configuration form for a command named 'curl'. It has fields for 'Command\*' (containing 'curl'), 'Arguments' (containing '-O saveToFile -u user:password http://targeturl'), and a note 'Enter each argument on a new line'. On the right, there is a 'Lookup Commands' panel showing the results for 'curl'. It lists 'curl' with a description 'Download from a protected Url and saveToFile' and an author 'author: Go Team'. There is also a 'more info' link.

The above screenshot resulted from the command below:

```
<!--
name: curl
description: Download from a protected Url and saveToFile
author: Go Team
authorinfo: http://support.thoughtworks.com/categories/20002778-go-community-support
keywords: curl, http, download, wget
moreinfo: http://curl.haxx.se/docs/manual.html
-->
<exec command="curl">
<arg>-o</arg>
<arg>saveToFile</arg>
<arg>-u</arg>
<arg>user:password</arg>
<arg>http://targeturl</arg>
</exec>
```

This is an example of valid command syntax. The command attribute is mandatory. No other attributes are valid. Zero or more arg child elements can be specified. No other child elements are allowed. One command file may only contain one command.

Please refer the [README](#) for full command and documentation syntax.

When you lookup a command, the following logic is used to sort the resulting suggestions:

1. Exact matches of name in command documentation (or filename if name missing)
2. Partial starts-with matches of name in command documentation (or filename if name missing)
3. Exact matches of keywords in command documentation

Within each category, the sorting is alphabetical.

## Contributing Commands

---

We welcome commands contributed by users. Simply,

1. [fork](#) this [GitHub repo](#)
2. Clone it locally
3. Commit and push your change
4. Send us a [pull request](#)

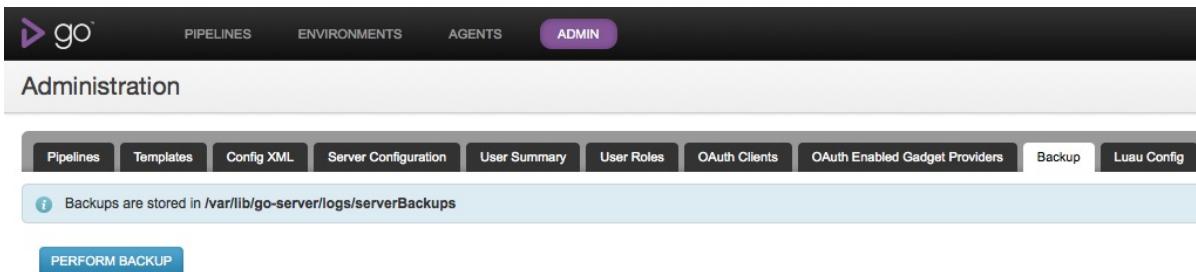
Accepted commands will be bundled into the next release.

# Backup Go Server

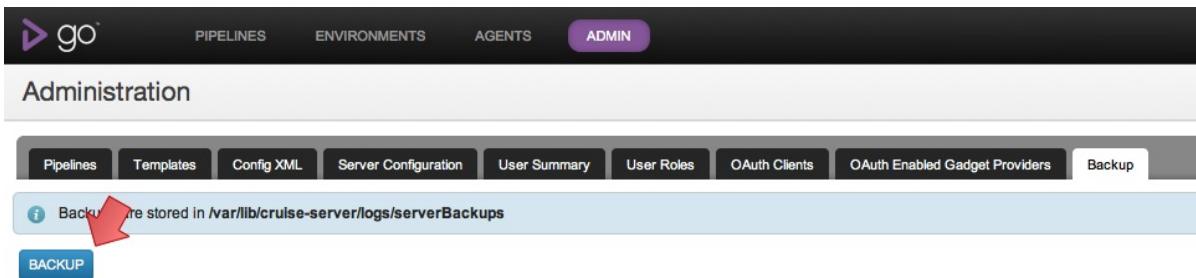
You can use Go's administration interface to perform an One-Click Backup of Go. You can also perform the backup [using the API](#).

## Steps to initiate backup

- On the Go Administration page, click on the Backup tab.



- Click on "BACKUP"



- Click "PROCEED"
- Backup time is proportional to the database and configuration size. We suggest you backup Go when the Go Server is idle. Users who are logged into the Go Dashboard will be redirected to a maintenance page during the backup. On backup completion they will be redirected to the page they were on.

## What is backed up?

The backup will be performed into the

{ARTIFACT\_REPOSITORY\_LOCATION}/serverBackups directory.

{ARTIFACT\_REPOSITORY\_LOCATION} for your server can be found as mentioned [here](#).

The backup directory will be named **backup\_{TIMESTAMP}** where the {TIMESTAMP} is the time when the backup was initiated.

- Database – This is in a zip called **db.zip**. The zip has a single DB file called

### cruise.h2.db

- Configuration – This is in a zip called **config-dir.zip** . This zip contains the XML configuration, Jetty server configuration, Keystores and all other Go's internal configurations.
- XML Configuration Version Repo – This is in a zip called **config-repo.zip** . This zip contains the Git repository of the XML configuration file.
- Go version – This is a file called **version.txt** . This file contains the version of the Go server when the backup was initiated

## What is not backed up?

Note: Please refer to the [this](#) page to see what the {SERVER\_INSTALLATION\_DIR} location is on different platforms.

The following are not backed up as a part of the Go backup process. Please ensure that these are manually backed up regularly.

- Artifacts - Please refer to [this section](#) to find out how to deal with artifacts
- Test Reporting Data - This is found at the location **{SERVER\_INSTALLATION\_DIR}/db/shine** . This contains the data used in the Failed Test History reporting
- Environment Variables - On Windows the environment variables that might be set for the user and on Linux the changes made to **/etc/default/go-server** are not backed up.
- Log Files

## Strategy to backup Artifacts and Test Reporting Data

Artifacts and the Test Reporting Data keep getting new files and directories added to them. So, it is a good idea to use **rsync** to copy the contents of these two into a backup location.

*For Instance:* Lets say you have a copy of all the files till 12-02-2012 in a location. On 20-02-2012, you can do something like:

```
rsync -avzP {ARTIFACT_LOCATION} {BACKUP_LOCATION}
```

This makes sure that only the files and directories that got newly added will be synced to the {BACKUP\_LOCATION} and not the entire contents.

# Restoring Go using backup

Note: Please refer to the [this](#) page to see what the `{SERVER_INSTALLATION_DIR}` location is on different platforms.

The restoration process is not automated and needs to be done manually. Please refer to the previous sections about the contents of the backup.

## Steps to restore

- In order to restore the Go server from a backup, the server must first be stopped. Make sure the process is completely dead before starting the restoration.
- Choose the backup directory that you want to restore from.

**Note: You cannot restore from a backup whose version is bigger than the version of the Go server being used.**

*For example:* If the backup is from version 12.3 and the server installation is of version 12.2, the restoration might not work. You can check the version of the backup from the `version.txt` file.

- You might want to keep a copy of all the files and directories that are involved in restoration. This will help in troubleshooting if there was a problem. Following this, make sure all the destination directories mentioned in the following steps are empty.

*For example:* Before restoring the Database, make sure the

`{SERVER_INSTALLATION_DIR}/db/h2db` is backed up and the directory is emptied.

- Database – Unzip the `db.zip` found in the backup directory. Unzip will create a file called `cruise.h2.db` . Copy this file to the directory `{SERVER_INSTALLATION_DIR}/db/h2db` .
- Configuration - Unzip the `config-dir.zip` into a temp directory. Copy all the files from this directory to `{SERVER_INSTALLATION_DIR}/config` directory on Windows and Mac or `/etc/go` on Linux and Solaris.
- Configuration History - Unzip the `config-repo.zip` into temp directory. Recursively copy all the contents from this directory to `{SERVER_INSTALLATION_DIR}/db/config.git` .
- Make sure the ownership of all the files that are restored are the same as the user running the Go server.

*For example:* Make sure you run a "chown -R go:go `{SERVER_INSTALLATION_DIR}/db/h2db`" after Database restoration.
- Start the Go server



# **INTEGRATING**

---

# **GO**

---

# **WITH OTHER TOOLS**

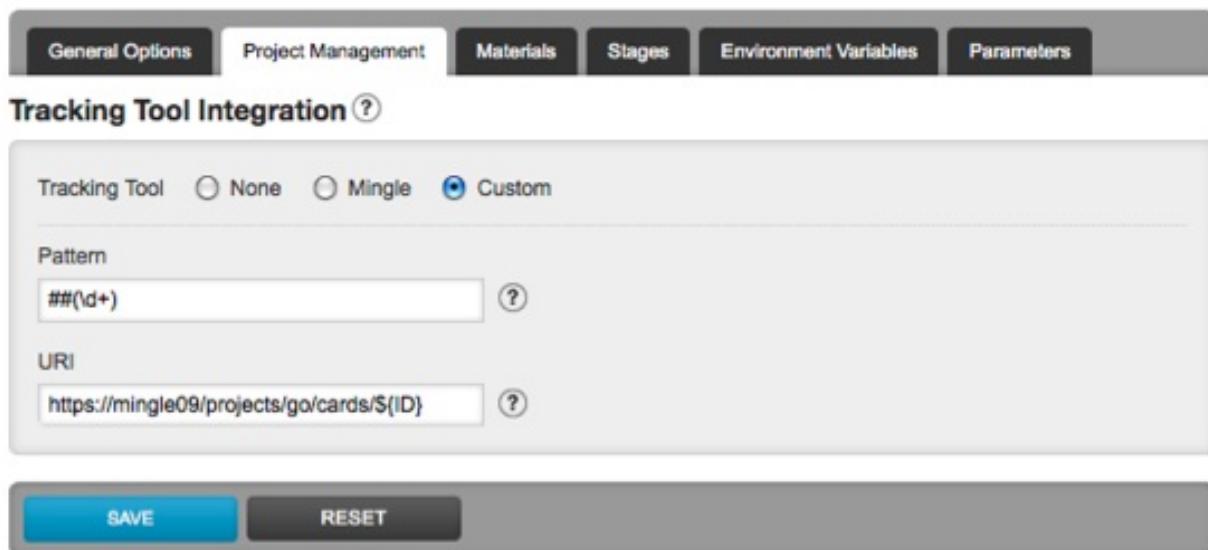
---

# Integration with external tools

## Integration with bug tracking and story management tools

Go allows you to link your commit messages with bug tracking and story management tools which are web applications.

The following shows an example of tracking cards on Mingle by card numbers starting with a "#" (e.g: #4618):



The pattern has an extra "#" because it is an escape character for a parameter. [More information...](#)

For power users, here's how you'd do it from Config XML:

```
<pipeline name="mypipeline">
  <trackingtool link="http://mingle.thoughtworks.com/go/\${ID}" regex="#\#\{id\+}" />
  ...
</pipeline>
```

Now, when a user commits code with comments like "#4618 - Make sure the TrackingToolConfig and...", Go detects the modifications and automatically links story/bug numbers to the configured tracking tool:

**494**

revision: 6573f8b20841...

1 day ago

Triggered by changes



## Monitoring your builds with client tools

Go allows you to monitor your builds with CCTray through the url

['http://\[your\\_go\\_server\]:8153/go/cctray.xml'](http://[your_go_server]:8153/go/cctray.xml).

If you have switched on security in your configuration you can use BASIC authentication to connect. In many cases you can do this by including the username and password in the URL. The username and password should not include special characters.

For example:

```
http://[username] : [password]@[your_go_server]:8153/go/cctray.xml
```

From 12.3, the CCTray feed includes a message tag with the "breakers" attribute that includes names of users who made the commit that broke the pipeline. Most CCTray clients support this message tag.

# Gadgets

Go provides a Pipeline gadget that enables you to see the latest status of the pipeline within any [OpenSocial](#) container like iGoogle, Mingle, etc.

The gadget is available at <https://<your-go-server>:8154/go/gadgets/pipeline.xml>

A sample of what the gadget will look like:



## Also see

- [Integrating Go with Mingle - an overview](#)

# Mingle Integration

---

Go integrates with [Mingle, the Agile Project Management tool](#) from ThoughtWorks Studios. This allows the users of Go and Mingle to surround their data with additional context, increasing the value of the information.

Go's [Mingle card activity gadget](#) allows users to see, in the context of Go, the new activity in a pipeline in terms of the Mingle cards that were worked on in that period. This card activity can reflect information about your project, such as which features were just deployed to production or which features require testing. To enable this integration, please follow the instructions for [configuring the display of Mingle gadgets within Go](#).

## Supported versions

To display a Go pipeline status gadget in Mingle, Mingle must be version 3.3 or greater and Go must be version 2.1 or greater.

To display a Mingle card activity gadget in Go, Mingle must be version 3.3 or greater and Go must be version 2.2 or greater.

## Integration technologies

---

### OpenSocial gadgets

ThoughtWorks Studios products use the [OpenSocial](#) gadget stack to provide UI integrations. This provides a secure, standards-based platform for displaying one application's content in another application. Part of this infrastructure is that Go is itself an OpenSocial gadget rendering server. In general, a gadget is simply a piece of content from another application, such as showing a Mingle card activity table inside of Go. If you use iGoogle and have customized your home page, you have worked with OpenSocial gadgets.

Most OpenSocial gadgets are publicly available content that do not require authentication and authorization. However, some gadgets, such as those published by the ThoughtWorks Studios products Go and Mingle, do require authorization. To accomplish this, Go's gadget rendering server supports OAuth 2.0 services.

Enabling Go for OAuth 2.0 enabled gadgets does require the Go administrator to take

[extra configuration steps](#).

If you are simply looking to configure the display of Mingle card activity gadgets in Go, please skip straight to the [instructions that are specific to showing Mingle gadgets in Go](#).

A gadget rendering server with OAuth 2.0 capabilities similar to what Go provides would be capable of showing ThoughtWorks Studios gadgets. That is, if iGoogle were to start supporting OAuth 2.0 in conjunction with its gadget support, and your Go instance was on a public server, it would be possible to display ThoughtWorks Studios gadgets on your home page. This is not currently possible so look for more on this from us in the future.

## OAuth 2.0

OAuth is an open-standard that enables a user to share private protected resources (e.g. photos or financial records) which she stores on one site (or application) with another one of her commonly used sites or applications without asking her to share any passwords between the two sites. OAuth is quickly becoming a widely adopted standard; for example, Yahoo, Facebook and Twitter have all adopted OAuth.

In the context of ThoughtWorks Studios applications, the application data is the private protected resource. For example, if someone had configured Go and Mingle integration, a Go user will only be allowed to see Mingle card information in Go that he would normally be allowed to see in Mingle. That is, when Go shows Mingle data in its pages, the Mingle authorization rules are not relaxed to allow all members of that Go pipeline group to automatically see the card activity for any Mingle project, rather he will only be able to see data from Mingle projects for which he has access. The same can be said when Mingle shows Go data in its pages. It should also be possible to maintain this principle when showing Go or Mingle content in a 3rd party, non-ThoughtWorks Studios context.

In order to make this work, Go and Mingle - and in the future all Studios server-based applications - use OAuth 2.0 (v9) as a means of allowing a user of one application to establish his identity in the other application, while also granting the appropriate data access for that user. Both applications are OAuth providers and both applications run a gadget rendering server capable of acting as an OAuth client. Currently, OAuth is only used for gadget-related communication, but we plan on expanding what data can be made available via OAuth in the future.

Below is a series of movies we've made that explain how OAuth works. Part 1 covers the basics and is likely enough for most users. Parts 2-4 get into the more technical details

of how the protocol works.

## Also see

- [Integrating Go with Mingle - an overview](#)
- [Reference for Mingle card activity gadget](#)
- [What is OAuth?](#)
- [What is OpenSocial?](#)

# Displaying Mingle gadgets in Go

---

Before a user can use the [Mingle card activity gadget](#) to display Mingle card activity for pipelines in Go, both the Mingle and Go administrators must do a bit of configuration. Mingle must be configured to be an OAuth provider. Go must be configured as an OAuth-capable gadget rendering server. OAuth trust, by way of client ID and client secret, must be established. These configuration steps, as well as a troubleshooting guide, are provided on this page.

A big part of the OAuth protocol depends upon Go and Mingle being configured with some special URLs that match exactly. And these URLs must be HTTPS endpoints. The Go administrator should double check that a secure site URL has been properly specified, via either the siteURL or secureSiteURL attributes of the server element, in the [Go configuration XML](#). The Mingle administrator should make a similar check of his Mingle configuration (although the mechanism for specifying site URLs differs in Mingle).

## Step 1 - Configure Mingle as OAuth 2.0 Provider

---

A Go user is only allowed to see Mingle card activity in Go that he would normally be allowed to see in Mingle. That is, when Go shows Mingle data in its pages, the Mingle authorization rules are not relaxed to allow all members of that Go pipeline group to automatically see the card activity. In order to make this work, Go and Mingle use OAuth 2.0 (v9) as a means of allowing the Go user to establish his identity in Mingle.

The first step of allowing Go and Mingle to use OAuth for this gadget is to configure Mingle as an OAuth 2.0 (v9) Provider. This step must be performed by a Mingle administrator.

[What is OAuth?](#)

### 1.1 - Capture Go OAuth Redirect URI

Before configuring Mingle, a bit of information must be captured from Go: the Go OAuth Redirect URI. If the Mingle administrator and Go administrator are not the same person, the Mingle admin will need to ask the Go admin for this piece of information. In Go, logged in as an administrator, navigate to the "Admin > OAuth Enabled Gadget

Providers" tab. On this page, in the blue info box, you'll see the OAuth Redirect URI. Copy and paste this URI to a scratch pad for use in the next step. Below is an example screenshot of Go displaying its OAuth Redirect URI. (Please do not attempt to derive your own redirect URL from this screenshot.)

## Details for this server as an OAuth client



Provide this information to your OAuth provider

**OAuth Redirect URI:** <https://go.example.com:9154/go/gadgets/oauthcallback>

## 1.2 - Create OAuth client entry in Mingle

In Mingle, logged in as an administrator, go to the home page that lists all projects. In the administration menu at the top of the page, click the 'Manage OAuth clients' link. Click on the "Add Client" button at the bottom of the page to create the new entry in Mingle allowing Go as an OAuth client. In the first field, enter a description of the OAuth client, most likely something like "Go" and in the second field, OAuth Redirect URI, enter the Go URL you captured in the previous step.

### New OAuth client

Name: *(The name of the client which will be accessing data from this server.)*

Go

OAuth Redirect URI: *(The redirect URI provided by the client.)*

<https://go.example.com:9154/go/gadgets/oauthcallback>

SUBMIT

CANCEL

Click the Submit button and note that Mingle has generated "Client ID" and "Client Secret" fields. Below is a listing similar to what you will see after you have successfully created the entry for a new OAuth client to Mingle. Copy and paste these values to a scratch pad for the next section. You will also need to copy the Authorization URL (in the blue info box) on this page. If the Mingle administrator and Go administrator are not the same people, the Mingle administrator will need to securely communicate these values to the Go administrator.

OAuth clients configured for this server

Name	OAuth Redirect URI	OAuth Client Credentials
Go	https://go.example.com:9154/go/gadgets/oauthcallback	<b>Client ID:</b> e9b3de7604ea52c375099205f332eb49e0122 <b>Client Secret:</b> 068896e1aa407b5daf53eb44dfc0871fd3e29a

## Step 2 - Configure Go to render gadgets from Mingle

In Mingle, we have just created an entry for a new OAuth client. Go is that client. You must now configure the Go half of that trust relationship. In Go, logged in as an administrator, navigate to the "Admin > OAuth Enabled Gadget Providers" tab. Click the 'Add Provider' button to create the new entry. Enter something along the lines of "Mingle" for the Name. Be sure that whatever you enter for Name is something that your users will recognize. For the OAuth Authorize URL, OAuth Client ID, and OAuth Client Secret fields enter the exact values you captured above in step 1.2.

### New OAuth enabled gadget provider

Name: (*The name of the service provider, i.e. the application whose data Go will be accessing*)

Mingle

OAuth Authorization URL: (*The URL of the OAuth Authorization server*)

https://mingle.example.com:8002/oauth/authorize

OAuth Client ID: (*The ID assigned to Go by the service provider*)

39205f332eb49e01229d2906f72ca1eb6a541bce1592d

OAuth Client Secret: (*The secret assigned to Go by the service provider*)

53eb44dfc0871fd3e29ade38a1d2c86a1ca1dba26f53bf

SUBMIT

CANCEL

Click submit to save the new entry. If you see the "Gadget provider was successfully created" message you can move on to Step 3.

However, if you see an error message like the one below, indicating that the certificate offered by the server is not trusted, you must complete one more action in this Step.



The certificate offered by the server at mingle.example.com:7071 is not trusted.

If you see the above message, you will also see a new section of text on the "New OAuth enabled gadget provider" page on which you have been working. This new section of text is the HTTPS certificate provided by Mingle to facilitate secure communication between Go and Mingle. You need to configure Go to trust this certificate. This piece of configuration is simply ticking the "Accept this configuration" checkbox and re-clicking "Submit." Go should now be correctly configured to retrieve gadget content from Mingle.

#### Certificate Details:

##### Issued to:

Common Name (CN)	David
Organization (O)	TW
Organizational Unit (OU)	TW
Serial Number	4CD05DDA

##### Issued by:

Common Name (CN)	David
Organization (O)	TW
Organizational Unit (OU)	TW

##### Validity:

Issued On	Tue Nov 02 18:52:10 UTC 2010
Expires On	Mon Jan 31 18:52:10 UTC 2011

##### Fingerprints:

SHA1 fingerprint	56:12:0D:78:17:6C:DD:10:E6:F4:AA:EE:78:72:DD:92:16:D1:3D:37
MD5 fingerprint	72:73:7B:A3:1F:3D:27:F2:32:F4:31:8A:05:A0:84:5F

Accept this certificate

SUBMIT

CANCEL

## Step 3 - Verify

The final step is to verify by testing an integration. Go to the reference for the [Mingle card activity gadget](#) and see if you can configure a Mingle card activity gadget for one of your Go pipelines. If the integration fails, please read through the troubleshooting section below.

## Troubleshooting

As you are configuring an integration, something might go wrong. Here are some of the

more common issues seen with this integration.

## Gadget does not render user authorization message... instead you see 'BAD\_OAUTH\_CONFIGURATION: There is no OAuth enabled gadget provider...

If users are seeing the error below rather than an opportunity to authorize the gadget provider to send Mingle data on your behalf there are two possible problems.

 OAuth error (BAD\_OAUTH\_CONFIGURATION): There is no registered OAuth enabled gadget provider for this gadget. Please contact your Go administrator for details.

The first possibility is that there simply is not a configured gadget provider. That is, a Go user attempted to use the Mingle card activity gadget without the Go administrator having properly configured the Mingle entry on the OAuth enabled gadget providers page.

The second possibility is that the Mingle URL or Mingle project identifier specified in the Mingle configuration (accessible in the Project Management section of Pipeline Administration) does not match the url specified for the value of the gadget provider's OAuth Authorization URL on the OAuth enabled gadget providers page. Please make sure that these values match.

## Authorization popup shows gadget provider configuration error

If, on clicking on the Authorize link, the popup opens but shows this error:

 Error in OAuth enabled gadget provider configuration. Please contact your Go administrator to resolve this issue.

One possible cause is that there is a mismatch between the redirect URI displayed on the Go OAuth Enabled Gadget Providers page and the redirect URI the Mingle administrator entered while creating the OAuth Client entry for Go. Double check that the values are identical.

Another possible cause is that there is a mismatch between the OAuth Client ID displayed on the Mingle OAuth Clients page and the value that the Go administrator entered while creating the gadget provider entry for Mingle. Double check that the values are identical.

When this error is showing in the authorization popup take a look at the value in the

browser's address bar. There is most likely a fairly readable error code contained in the address that will reveal the specific configuration error.

## User clicks 'Yes' on authorization popup but gadget still not showing

If, upon clicking Yes in the authorization popup, the user sees the error below showing invalid client credentials, there is a mismatch between the OAuth Client Secret values. Double check that the OAuth Client Secret displayed on the Mingle OAuth Clients page is identical to the value that the Mingle administrator entered while creating the gadget provider entry for Go. Note that this error is not a reference to the user's credentials, but to the OAuth trust between Go and Mingle.



OAuth error (invalid-client-credentials)

## Also see

- [Integrating Go with Mingle - an overview](#)
- [Reference for Mingle card activity gadget](#)
- [What is OAuth?](#)
- [What is OpenSocial?](#)

# Mingle Card Activity Gadget

Go's Mingle card activity gadget allows users to see the new activity in a pipeline in terms of the Mingle cards that were worked on in that period. This card activity can reflect information about your project, such as which features were just deployed to production or which features require testing.

For this feature to work, the Go and Mingle administrators must first [configure the display of Mingle gadgets in Go](#).

## Configuring Mingle Card Activity for a Go pipeline

Navigate to the Administration page for the pipeline for which you would like to view card activity.

Open the 'Project Management' section for the pipeline and select 'Mingle' for the 'Tracking Tool Integration'.

The screenshot shows the 'Project Management' tab selected in the top navigation bar. In the 'Tracking Tool Integration' section, the 'Mingle' radio button is selected and highlighted with a red box. Below the radio buttons are three input fields: 'URI', 'Project Identifier', and 'MQL Grouping Conditions', each with a help icon (question mark) to its right. At the bottom of the form are two buttons: 'SAVE' (in blue) and 'RESET' (in grey).

There are three fields used by the card activity feature:

- **URI** - Required field. The base URI for the Mingle instance that hosts your Mingle project.
- **Project Identifier** - Required field. The identifier for your Mingle project. This identifier can be found in the 'Basic Information' section under the 'Project admin' tab for your Mingle project.
- **MQL Grouping Conditions** - Optional field. MQL snippet used to provide additional information regarding the cards that appear in your card activity table. This MQL is used to determine whether the cards are in a particular state, such as "Greater than In Development." If this is supplied, the cards will be grouped into cards that currently meet the MQL condition, cards that once met the conditions but no longer do, and those that have never met the conditions.

Below is an example configuration for the card activity feature:

The screenshot shows a configuration interface for tracking tool integration. At the top, there are tabs: General Options (selected), Project Management, Materials, Stages, Environment Variables, and Parameters. Below the tabs, the title is "Tracking Tool Integration". Under "Tracking Tool", the "Mingle" radio button is selected. The "URI" field contains "https://mingle.example.com:8443/" and the "Project Identifier" field contains "example\_project". Under "MQL Grouping Conditions", the value is "'Story Status' > 'Development in Progress'". At the bottom, there are "SAVE" and "RESET" buttons.

## Accessing Mingle Card Activity in Go

There are several locations from where Mingle card activity can be accessed. These include:

- Pipelines Dashboard page
- Pipeline instance details page
- Environments page
- Stage History section on the Materials page for a pipeline

On each of these pages, a 'Compare' link will be displayed for each pipeline or pipeline instance. For example, the screenshot below depicts the 'Compare' link as displayed on the Pipelines Dashboard page.



## Card Activity information

For cards to be included in the card activity list, the commit messages must include the card number in the following format: #card\_number (e.g. #412). Do not put a space between the # and the card number.

### With MQL Grouping Conditions

When MQL Grouping Conditions are used, the cards are listed in groups according to whether they:

Icon	Meaning
✓	Currently meets the conditions
!	Did meet the conditions, but no longer does
✗	Never met the conditions

Below is an example of what the card activity will look like when MQL grouping conditions are supplied. Cards currently meeting the conditions will be shown at the top

of the list.

Grouping Conditions: 'Story Status' > 'Development in Progress'			
	#	Name	Met Grouping Conditions
✓	95	Subscribe to RSS feed	1 day ago
✓	14	Sign out of my online mailbox	1 day ago
!	10	Save sent messages in an online 'sent items' box	11 days ago (until 15 Apr 2011)
✗	17	Reply to email	never

## Without MQL Grouping Conditions

When MQL Grouping Conditions are not provided, the cards are listed in the order in which the commits were made against them.

Changes		Card Activity
#	Name	
10	Save sent messages in an online 'sent items' box	
17	Reply to email	
95	Subscribe to RSS feed	
14	Sign out of my online mailbox	

Card activity reflects the "live" state of Mingle, at the time you are viewing this page. That is, the card activity shown is not a snapshot of Mingle data from the time of pipeline execution or deployment. As changes are made to cards in Mingle, this page will reflect the latest card activity.

## Also see

- [Integrating Go with Mingle - an overview](#)
- [Reference for Mingle card activity gadget](#)
- [What is OAuth?](#)
- [What is OpenSocial?](#)

# **APIs**

---

## **IN GO**

---

# Go API

---

## Introduction

---

RESTful APIs offer a convenient way to integrate networked systems. Go provides a RESTful API through which you can access and manipulate various kinds of resources. The following sections will walk you through the various resources and the operations that are supported on each of them.

## Go API features

The Go API provides access to various kinds of resources in Go over HTTP. The standard HTTP operation of GET, POST and PUT and DELETE are used to support read, create and upload operations on them. All Go APIs use HTTP basic authentication. Hence performing a GET operation for Artifacts would be the equivalent of getting that resource, performing a PUT would be the equivalent of updating that artifact; and so on.

## Main APIs

The Go API documented here is a work in progress. Future versions may change this API.

Go's main APIs are listed below:

- [Pipeline Group API](#)
- [Pipeline API](#)
- [Stages API](#)
- [Job API](#)
- [Agent API](#)
- [Materials API](#)
- [Configuration API](#)
- [Artifacts API](#)
- [Users API](#)
- [Backup API](#)
- [Properties API](#)
- [Feeds API](#)
- [Command Repo API](#)

# Pipeline Group API

## Introduction

The Go API documented here is a work in progress. Future versions may change this API.

## Config listing API

This API allows you to list Pipeline groups, Pipelines in each group, Material & Stage in each pipeline in JSON format. This API is built primarily to aid rendering of VSM for config. Hence only the information required for that is exposed.

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/config/pipeline_groups">http://[server]/go/api/config/pipeline_groups</a>	GET	no parameters	List all Pipeline Groups.

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

The pipeline configuration looks like:

```
<pipelines group="first">
  <pipeline name="foo" labeltemplate="foo-${COUNT}" isLocked="true">
    <materials>
      <hg url="http://10.22.12.2:8000" materialName="hg_material" />
    </materials>
    <stage name="DEV">
      <jobs>
        <job name="UnitTest">
          <tasks>
            <ant target="ut" />
          </tasks>
        </job>
      </jobs>
    </stage>
  </pipeline>
</pipelines>
```

```

        </job>
    </jobs>
</stage>
<stage name="UATTest">
    <jobs>
        <job name="UAT">
            <tasks>
                <ant target="all-UAT" />
            </tasks>
            <artifacts>
                <artifact src="target" dest="pkg/foo.war" />
            </artifacts>
        </job>
    </jobs>
</stage>
</pipeline>
</pipelines>

```

The following command produces output specified below:

```
curl -u admin:badger http://goserver.com:8153/go/api/config/pipeline_groups
```

```
[
{
  "pipelines": [
    {
      "stages": [
        {
          "name": "DEV"
        },
        {
          "name": "UATTest"
        }
      ],
      "name": "foo",
      "materials": [
        {
          "description": "URL: http://10.22.12.2:8000",
          "fingerprint": "64987f67c407020dfd6badf4975421428aa5d044e0b14b30862662941",
          "type": "Mercurial"
        }
      ],
      "label": "foo-${COUNT}"
    }
  ],
  "name": "first"
}
]
```

# Pipeline API

---

## Introduction

---

The Go API documented here is a work in progress. Future versions may change this API.

## Scheduling pipelines

---

You can specify particular versions of the materials to use for the new pipeline. If you do not specify a particular revision for a material, Go will use the latest.

To choose which revision to use for a material it must have a **materialName** defined. By default the materialName of an upstream pipeline is the name of that pipeline. You can override this and specify a materialName, and then use this in the following APIs.

You can also parametrize your deployment script with environment variables at the time of triggering a pipeline. You can specify the value for any of the environment variables specified in the configuration file. This value will get carried all the way through to the relevant jobs. You can override the value of an environment variables specified at the environment, pipeline, stage or job level(in the configuration file) for that pipeline.

If a new value for an environment variable is not provided at the time of triggering the pipeline, then the values specified in the configuration file for this pipeline will be used.

URL format: `http://[server]:8153/go/api/pipelines/[pipeline]/schedule`

HTTP Verb	Data
POST	no parameters

POST	materials[svn_material]=3456
POST	materials[repo-name:pkg-name]=gcc-4.4.7-3.el6.x86_64
POST	materials[svn_material]=3456&materials[upstream_foo]=upstream_foo/2/dist/1
POST	materials[svn_material]=3456&materials[my-upstream-pipeline-name]=upstream_bar/2/dist/1

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/> .
- We assume security has been switched on, and that there is a user named **admin** with the password **badger** .

The upstream pipeline (which is a material for 'foo') looks like:

```

<pipeline name="upstream_foo" labeltemplate="upstream_foo-1.0-${{COUNT}}">
    <material>
        <svn url="..."/>
    </material>
    <stage name="Dist">
        <job name="dist">
            <tasks>
                <ant target="dist"/>
            </tasks>
        </job>
    </stage>
</pipeline>
.....
<pipeline name="upstream_bar" labeltemplate="upstream_bar-1.2-${{COUNT}}">
    ...

```

And the pipeline configuration looks like:

```

<pipeline name="foo" labeltemplate="foo-1.0-${COUNT}">
    <environmentvariables>
        <variable name="MACHINE_IP"><value>10.22.12.2</value></variable>
        <variable name="PASSWORD" secure="true"><encryptedValue>${ENCRYPTED_PASSWORD}</encryptedValue></variable>
    </environmentvariables>
    <material>
        <svn url="http://thoughtworks.com:8080" materialName="src">
            <svn url="http://thoughtworks.com:8080" materialName="src">
                <pipeline pipelineName="upstream_foo" stageName="Dist"/>
                <pipeline pipelineName="upstream_bar" stageName="Install" />
                <hg url="http://10.22.12.2:8000" materialName="hg_material" />
            </svn>
        </material>
    <stage name="DEV">
        <environmentvariables>
            <variable name="MACHINE_IP">10.22.2.12</variable>
        </environmentvariables>
        <job name="UnitTest">
            <environmentvariables>
                <variable name="TLB_TMP_DIR">C:\tlb_tmp_dir</variable>
            </environmentvariables>
            <tasks>
                <ant target="ut"/>
            </tasks>
            <artifacts>
                <artifact src="coverage" dest="coveragereport.html"/>
            </artifacts>
        </job>
    </stage>
    <stage name="UATTest">
        <job name="UAT">
            <tasks>
                <ant target="all-UAT"/>
            </tasks>
            <artifacts>
                <artifact src="report" dest="UATreport.html"/>
            </artifacts>
        </job>
    </stage>
</pipeline>

```

```
<artifact src="target" dest="pkg/foo.war"/>
</artifacts>
</job>
</stage>
</pipeline>
```

If you want to trigger a new instance of the pipeline with the latest of all materials

```
curl -u admin:badger -d "" http://goserver.com:8153/go/api/pipelines/foo/schedule
```

If you want to trigger a new instance of the pipeline 'foo' with revision '3456' of your svn repository and instance 'upstream\_foo/1/dist/2' of the upstream pipeline

```
curl -u admin:badger -d "materials[svn_material]=3456&materials[upstream_foo]=upstream_foo/1/dist/2" http://goserver.com:8153/go/api/pipelines/foo/schedule
```

If you want to trigger a new instance of the pipeline 'foo' with revision '3456' of your svn repository and instance 'upstream\_bar/1/installers/2' of the upstream pipeline

```
curl -u admin:badger -d "materials[svn_material]=3456&materials[my-upstream-pipeline]=upstream_bar/1/installers/2" http://goserver.com:8153/go/api/pipelines/foo/schedule
```

You can also use the following form, passing the materials as part of the URL

```
curl -u admin:badger -d "materials[svn_material]=3456&materials[upstream_foo]=upstream_foo/1/dist/2" http://goserver.com:8153/go/api/pipelines/foo/schedule
```

If you want to trigger a new instance of the pipeline 'foo' with revision '3456' of your svn repository and parametrize the environment variable MACHINE\_IP with new value '10.21.2.2' for this specific run

```
curl -u admin:badger -d "materials[svn_material]=3456&variables[MACHINE_IP]=10.21.2.2" http://goserver.com:8153/go/api/pipelines/foo/schedule
```

If you want to trigger a new instance of the pipeline with the latest of all materials and

parametrize the environment variable MACHINE\_IP with new value '10.21.2.2' for this specific run

```
curl -u admin:badger -d "variables[MACHINE_IP]=10.21.2.2&variables[TLB_TMP_DIR]=C:\temp\test"
```

- Similar to overriding variables, you can override secure variables while triggering a new instance of the pipeline

If you want to trigger a new instance of the pipeline with the latest of all materials and parametrize the secure variable PASSWORD with a new value 'new\_password' for this specific run

```
curl -u admin:badger -d "secure_variables[PASSWORD]=new_password" http://goserver.com/api/pipelines/test/trigger
```

## Releasing a pipeline lock

This API allows you to release a lock on a pipeline so that you can start up a new instance without having to wait for the earlier instance to finish.

A pipeline lock can only be released when:

- A locked pipeline has stopped because of a failed stage
- A locked pipeline has stopped because of a canceled stage
- A locked pipeline is waiting for a manual stage (i.e. a stage requiring manual approval)

URL format	HTTP Verb	Data	Example
<a href="http://[server]/go/api/pipelines/[pipeline]/releaseLock">http://[server]/go/api/pipelines/[pipeline]/releaseLock</a>	POST	no parameters	Flockstop

### Response Codes

HTTP response code	Explanation
200	Success

200	pipeline lock released for [pipeline]
404	[pipeline] is does not exist
406	no lock exists within the pipeline configuration for [pipeline]
406	lock exists within the pipeline configuration but no pipeline instance is currently in progress
406	locked pipeline instance is currently running (one of the stages is in progress)
401	user does not have operate permission on the pipeline

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

The pipeline configuration looks like:

```

<pipeline name="foo" labeltemplate="foo-1.0-${COUNT}" isLocked="true">
    <material>
        <hg url="http://10.22.12.2:8000" materialName ="hg_material">
    </material>
    <stage name="DEV">
        <job name="UnitTest">
            <tasks>
                <ant target="ut"/>
            </tasks>
        </job>
    </stage>
    <stage name="UATTest">
        <job name="UAT">
            <tasks>
                <ant target="all-UAT"/>
            </tasks>
            <artifacts>
                <artifact src="target" dest="pkg/foo.war"/>
            </artifacts>
        </job>
    </stage>
</pipeline>

```

Let's say the "DEV" stage failed in an instance of pipeline "foo" . Run this command to release the lock:

```
curl -u admin:badger -d "" http://goserver.com:8153/go/api/pipelines/foo/releaseLock
```

## Pause a pipeline

API to pause a pipeline needs the following as input:

- Name of the pipeline.
- Reason for pausing the pipeline.

Security Note: The user invoking the API should have sufficient permission to operate on the pipeline.

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/pipelines/[pipeline]/pause">http://[server]/go/api/pipelines/[pipeline]/pause</a>	POST	pauseCause	Pauses specific pipeline with the given reason.

### Response Codes

HTTP response code	Explanation
200	[pipeline] paused with the given cause.
404	[pipeline] does not exist.
401	User does not have operate permission on the pipeline.

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

Assuming the pipeline configuration looks like:

```

<pipeline name="demo_pipeline" labeltemplate="demo_pipeline-1.0-%d">
    <material>
        <svn url="..."/>
    </material>
    <stage name="first_stage">
        <job name="first_job">
            <tasks>
                <ant target="run"/>
            </tasks>
        </job>
    </stage>
</pipeline>
....
```

Run this command to pause the pipeline:

```
curl -u admin:badger -d "pauseCause=take some rest" http://goserver.com:8153/go/api/pipelines/demo_pipeline/pause
```

## Unpause a pipeline

API to unpause a pipeline needs only the name of the pipeline as input.

**Security Note:** The user invoking the API should have sufficient permission to operate on the pipeline.

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/pipelines/[pipeline]/unpause">http://[server]/go/api/pipelines/[pipeline]/unpause</a>	POST	no parameters	Unpauses the specified pipeline.

### Response Codes

HTTP response code	Explanation
200	[pipeline] successfully unpause.
404	[pipeline] does not exist.
401	User does not have operate permission on the pipeline.

### Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

Assuming the pipeline configuration looks like:

```
<pipeline name="demo_pipeline" labeltemplate="demo_pipeline-1.0-%s">
  <material>
    <svn url="..." />
  </material>
  <stage name="first_stage">
    <job name="first_job">
      <tasks>
        <ant target="run" />
      </tasks>
    </job>
  </stage>
</pipeline>
....
```

Run this command to unpause the pipeline:

```
curl -u admin:badger -d "" http://goserver.com:8153/go/api/pipelines/demo_pipeline/unpause
```

## Pipeline Status

This API can be used to check if the APIs like "schedule pipeline", "release lock", "pause pipeline", "un-pause pipeline" etc. need to be & can be invoked.

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/pipelines/[pipeline]/status">http://[server]/go/api/pipelines/[pipeline]/status</a>	GET	no parameters	JSON contain information about paused, locked, scheduled pipelines

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

Assuming the pipeline configuration looks like:

```
<pipelines group="first">
  <pipeline name="foo" labeltemplate="foo-${COUNT}" isLocked="true">
    <materials>
      <hg url="http://10.22.12.2:8000" materialName="hg_material" />
    </materials>
    <stage name="DEV">
      <jobs>
        <job name="UnitTest">
          <tasks>
            <ant target="all-UAT" />
          </tasks>
        </job>
      </jobs>
    </stage>
  </pipeline>
</pipelines>
```

The following command produces output specified below:

```
curl -u admin:badger http://goserver.com:8153/go/api/pipelines/foo/status
```

```
{
  "locked": true,
  "paused": false,
  "schedulable": false
}
```

## Pipeline History

This API lists Pipeline instances in JSON format. API gives 10 instances at a time, sorted in reverse order. Supports pagination using offset which tells the API how many instances to skip. This API is built primarily to aid rendering pipeline history page. Hence

the information required for that is exposed.

URL format	HTTP Verb	Data
<a href="http://[server]/go/api/pipelines/[pipeline]/history/[offset]">http://[server]/go/api/pipelines/[pipeline]/history/[offset]</a>	GET	no parameters

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

Assuming the pipeline configuration looks like:

```
<pipelines group="first">
  <pipeline name="foo" labelTemplate="foo-${COUNT}" isLocked="true">
    <materials>
      <hg url="http://10.22.12.2:8000" materialName="hg_material" />
    </materials>
    <stage name="DEV">
      <jobs>
        <job name="UnitTest">
          <tasks>
            <ant target="all-UAT" />
          </tasks>
        </job>
      </jobs>
    </stage>
    <stage name="UATTest">
      <jobs>
        <job name="UAT">
          <tasks>
            <exec command="sleep">
              <arg>10000</arg>
              <runif status="passed" />
            </exec>
          </tasks>
          <artifacts>
            <artifact src="target" dest="pkg/foo.war" />
          </artifacts>
        </job>
      </jobs>
    </stage>
  </pipeline>
</pipelines>
```

The following command produces output specified below:

```
curl -u admin:badger http://goserver.com:8153/go/api/pipelines/foo/history/0
```

```
{
  "pipelines": [
    {
      "build_cause": {
        "approver": "anonymous",
        "material_revisions": [
          {
            "modifications": [
              {
                "modified_time": 1409664999000,
                "id": 1,
                "user_name": "Admin <test@test.com>",
                "comment": "comment",
                "revision": "6f75b392941c40666ae822045c064e0887ffd007"
              }
            ],
            "material": {
              "description": "URL: http://10.22.12.2:8000",
              "fingerprint": "64987f67c407020dfd6badf4975421428aa5d044e0b14b30862662",
              "type": "Mercurial",
              "id": 34
            },
            "changed": false
          }
        ],
        "trigger_forced": true,
        "trigger_message": "Forced by anonymous"
      },
      "name": "foo",
      "natural_order": 2,
      "can_run": false,
      "stages": [
        {
          "name": "DEV",
          "approved_by": "admin",
          "jobs": [
            {
              "name": "UnitTest",
              "result": "Failed",
              "state": "Completed",
              "id": 3,
              "scheduled_date": 1411456876262
            }
          ],
          "can_run": false,
          "approval_type": "success",
          "result": "Failed",
          "id": 3,
          "counter": "1",
          "operate_permission": true,
        }
      ]
    }
  ]
}
```

```
        "scheduled": true
    },
{
    "name": "UATTest",
    "approved_by": "admin",
    "jobs": [
        {
            "name": "UAT",
            "result": "Unknown",
            "state": "Building",
            "id": 4,
            "scheduled_date": 1411458820384
        }
    ],
    "can_run": false,
    "approval_type": "success",
    "result": "Unknown",
    "id": 4,
    "counter": "1",
    "operate_permission": true,
    "scheduled": true
},
],
"currently_locked": false,
"counter": 2,
"id": 2,
"preparing_to_schedule": false,
"lockable": false,
"can_unlock": false,
"label": "foo-2"
},
{
    "build_cause": {
        "approver": "",
        "material_revisions": [
            {
                "modifications": [
                    {
                        "modified_time": 1409664999000,
                        "id": 1,
                        "user_name": "Admin <test@test.com>",
                        "comment": "comment",
                        "revision": "6f75b392941c40666ae822045c064e0887ffd007"
                    }
                ],
                "material": {
                    "description": "URL: http://10.22.12.2:8000",
                    "fingerprint": "64987f67c407020dfd6badf4975421428aa5d044e0b14b30862662",
                    "type": "Mercurial",
                    "id": 34
                },
                "changed": true
            }
        ],
        "trigger_forced": false,
        "trigger_message": "modified by Admin <test@test.com>"
    },
    "name": "foo",
    "natural_order": 1,
```

```
"can_run": false,
"stages": [
  {
    "name": "DEV",
    "approved_by": "changes",
    "jobs": [
      {
        "name": "UnitTest",
        "result": "Passed",
        "state": "Completed",
        "id": 1,
        "scheduled_date": 1411456676119
      }
    ],
    "can_run": false,
    "approval_type": "success",
    "result": "Passed",
    "id": 1,
    "counter": "1",
    "operate_permission": true,
    "scheduled": true
  },
  {
    "name": "UATTest",
    "approved_by": "changes",
    "jobs": [
      {
        "name": "UAT",
        "result": "Failed",
        "state": "Completed",
        "id": 2,
        "scheduled_date": 1411456763411
      }
    ],
    "can_run": false,
    "approval_type": "success",
    "result": "Failed",
    "id": 2,
    "counter": "1",
    "operate_permission": true,
    "scheduled": true
  }
],
"currently_locked": false,
"counter": 1,
"id": 1,
"preparing_to_schedule": false,
"lockable": false,
"can_unlock": false,
"label": "foo-1"
},
],
"pagination": {
  "offset": 0,
  "total": 2,
  "page_size": 10
}
}
```



# Stages API

---

## Introduction

---

The Go API documented here is a work in progress. Future versions may change this API.

## Stage Cancellation API

---

This API provides the ability to cancel an active stage of a pipeline. The API needs the name of the pipeline and name of the stage to perform cancellation.

Security Note: The user invoking the API should have sufficient permission to operate on the pipeline.

URL format	HTTP Verb	Data
<code>http://[server]:8153/go/api/stages/[pipeline]/[stage]/cancel</code>	POST	no parameters

### Response Codes

HTTP response code	Explanation
200	given stage was successfully cancelled or the stage was not active.
404	given stage does not exist.
401	User does not have operate permission on the give stage.

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

Assuming the pipeline configuration looks like:

```
<pipeline name="demo_pipeline" labeltemplate="demo_pipeline-1.0-:1">
    <material>
        <svn url="..."/>
    </material>
    <stage name="first_stage">
        <job name="first_job">
            <tasks>
                <ant target="run"/>
            </tasks>
        </job>
    </stage>
</pipeline>
....
```

Run this command to cancel the stage of the pipeline:

```
curl -u admin:badger -d "" http://goserver.com:8153/go/api/stages/demo_pipeline/first
```

## Stage History

This API lists Stage instances in JSON format. API gives 10 instances at a time, sorted in reverse order. Supports pagination using offset which tells the API how many instances to skip. This API is built primarily to aid rendering Stage history widget. Hence the information required for that is exposed.

URL format	HTTP Verb	Data
<a href="http://[server]/go/api/stages/[pipeline]/[stage]/history/[offset]">http://[server]/go/api/stages/[pipeline]/[stage]/history/[offset]</a>	GET	no parameters

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger** .

Assuming the pipeline configuration looks like:

```
<pipelines group="first">
  <pipeline name="foo" labelTemplate="foo-${COUNT}" isLocked="true">
    <materials>
      <hg url="http://10.22.12.2:8000" materialName="hg_material" />
    </materials>
    <stage name="DEV">
      <jobs>
        <job name="UnitTest">
          <tasks>
            <ant target="all-UAT" />
          </tasks>
        </job>
      </jobs>
    </stage>
    <stage name="UATTest">
      <jobs>
        <job name="UAT">
          <tasks>
            <exec command="sleep">
              <arg>10000</arg>
              <runif status="passed" />
            </exec>
          </tasks>
          <artifacts>
            <artifact src="target" dest="pkg/foo.war" />
          </artifacts>
        </job>
      </jobs>
    </stage>
  </pipeline>
</pipelines>
```

The following command produces output specified below:

```
curl -u admin:badger http://goserver.com:8153/go/api/stages/foo/DEV/history/0
```

```
{
  "stages": [
    {
      "name": "DEV",
      "approved_by": "admin",
      "jobs": [
        {
          "name": "UnitTest",
          "result": "Failed",
          "state": "Completed",
          "id": 3,
          "scheduled_date": 1411456876262
        }
      ]
    }
  ]
}
```

```
],
  "pipeline_counter": 2,
  "pipeline_name": "foo",
  "approval_type": "success",
  "result": "Failed",
  "id": 3,
  "counter": "1"
},
{
  "name": "DEV",
  "approved_by": "changes",
  "jobs": [
    {
      "name": "UnitTest",
      "result": "Passed",
      "state": "Completed",
      "id": 1,
      "scheduled_date": 1411456676119
    }
  ],
  "pipeline_counter": 1,
  "pipeline_name": "foo",
  "approval_type": "success",
  "result": "Passed",
  "id": 1,
  "counter": "1"
}
],
"pagination": {
  "offset": 0,
  "total": 2,
  "page_size": 10
}
}
```

# Job API

---

## Introduction

---

The Go API documented here is a work in progress. Future versions may change this API.

## Scheduled Jobs

---

This api gives a list of all the current job instances which are scheduled but not yet assigned to any agent.

The XML output provides:

- Pipeline, stage and their counters for this job instance.
- Resources allotted to the job.
- Environments the job's pipeline belongs to.
- Environment Variables configured for the job.

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/jobs/scheduled.xml">http://[server]/go/api/jobs/scheduled.xml</a>	GET	no parameters	Get all scheduled jobs

## Examples

- We use curl,a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/> .
- We assume security has been switched on, and that there is a user named **admin** with the password **badger** .

Run this command to get the list of scheduled jobs:

```
curl -u admin:badger -d "" http://go-server.com:8153/go/api/jobs/scheduled.xml
```

Sample output is shown below:

```
<scheduledJobs>
  <job name="fresh.install.go" id="186225">
    <link rel="self" href="http://go-server:8153/go/tab/build/detail/auto-dep"
      <buildLocator>
        auto-deploy-testing-open-solaris/11/fresh-install/1/fresh.install.go
      </buildLocator>
    <environment>AutoDeploy-OpenSolaris</environment>
    <resources>
      <resource>
        <autodeploy >
      </resource>
    </resources>
    <environmentVariables>
      <variable name="TWIST_SERVER_PATH">/etc/go</variable>
      <variable name="TWIST_SERVER_CONFIG_PATH">/etc/go</variable>
      <variable name="TWIST_AGENT_PATH">/var/lib/go-agent</variable>
    </environmentVariables>
  </job>
  <job name="publish" id="285717">
    <link rel="self" href="http://go-server:8153/go/tab/build/detail/go-ec2-p
      <buildLocator>go-ec2-plugin/26/dist/1/publish</buildLocator>
    <environment>performance-ec2</environment>
    <resources>
      <resource>
        <deploy-agent>
      </resource>
    </resources>
  </job>
  <job name="upgrade" id="297092">
    <link rel="self" href="http://go-server:8153/go/tab/build/detail/upgrade_
      <buildLocator>upgrade_qa_server/15/upgrade/1/upgrade</buildLocator>
    <environment>UAT</environment>
    <resources>
      <resource>
        <UAT-Server>
      </resource>
    </resources>
  </job>
</scheduledJobs>
```

## Job History

This API lists Job instances in JSON format. API gives 10 instances at a time, sorted in reverse order. Supports pagination using offset which tells the API how many instances to skip. This API is built primarily to aid rendering Job history widget. Hence the information required for that is exposed.

URL format	HTTP Verb	Data
<code>http://[server]/go/api/jobs/[pipeline]/[stage]/[job]/history/[offset]</code>	GET	no parameters

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

Assuming the pipeline configuration looks like:

```
<pipelines group="first">
  <pipeline name="foo" labelTemplate="foo-${COUNT}" isLocked="true">
    <materials>
      <hg url="http://10.22.12.2:8000" materialName="hg_material" />
    </materials>
    <stage name="DEV">
      <jobs>
        <job name="UnitTest">
          <tasks>
            <ant target="all-UAT" />
          </tasks>
        </job>
      </jobs>
    </stage>
    <stage name="UATTest">
      <jobs>
        <job name="UAT">
          <tasks>
            <exec command="sleep">
              <arg>10000</arg>
              <runif status="passed" />
            </exec>
          </tasks>
          <artifacts>
            <artifact src="target" dest="pkg/foo.war" />
          </artifacts>
        </job>
      </jobs>
    </stage>
  </pipeline>
</pipelines>
```

The following command produces output specified below:

```
curl -u admin:badger http://goserver.com:8153/go/api/jobs/foo/DEV/UnitTest/history/0
```

```
{
  "jobs": [
    {
      "agent_uuid": "0794793b-5c1a-443f-b860-df480986586b",
      "name": "UnitTest",
      "job_state_transitions": [],
      "scheduled_date": 1411456876262,
      "pipeline_counter": 2,
      "rerun": false,
      "pipeline_name": "foo",
      "result": "Failed",
      "state": "Completed",
      "id": 3,
      "stage_counter": "1",
      "stage_name": "DEV"
    },
    {
      "agent_uuid": "0794793b-5c1a-443f-b860-df480986586b",
      "name": "UnitTest",
      "job_state_transitions": [],
      "scheduled_date": 1411456676119,
      "pipeline_counter": 1,
      "rerun": false,
      "pipeline_name": "foo",
      "result": "Passed",
      "state": "Completed",
      "id": 1,
      "stage_counter": "1",
      "stage_name": "DEV"
    }
  ],
  "pagination": {
    "offset": 0,
    "total": 2,
    "page_size": 10
  }
}
```

# Agent API

---

A collection of APIs to get information and do operations on agents. They allow Go administrators to provision and de-provision agents without needing to use the web interface.

## List Agents

---

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/agents">http://[server]/go/api/agents</a>	GET	no parameters	List all the agents of the Go Server in JSON format along with their information.

The following information about each of the agents is returned in the JSON response:

- `uuid`
- `buildLocator`
- `agent_name` : (Maps to "Agent Name" column of Agents tab)
- `ipAddress` : (Maps to "IP Address" column of Agents tab)
- `status` : (Maps to "Status" column of Agents tab)
- `sandbox` : (Maps to "Sandbox" column of Agents tab)
- `os` : (Maps to "OS" column of Agents tab)
- `free_space` : (Maps to "Free Space" column of Agents tab)
- `resources` : (Maps to "Resources" column of Agents tab)
- `environments` : (Maps to "Environments" column of Agents tab)

## Enable Agent

---

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/agents/[agent-uuid]/enable">http://[server]/go/api/agents/[agent-uuid]/enable</a>	POST	no parameters	Enable a disabled agent. Approve a pending agent.

## Disable Agent

URL format	HTTP Verb	Data	Explanation
<code>http://[server]/go/api/agents/[agent-uuid]/disable</code>	POST	no parameters	Disable an enabled/pending agent.

## Delete Agent

URL format	HTTP Verb	Data	Explanation
<code>http://[server]/go/api/agents/[agent-uuid]/delete</code>	POST	no parameters	Delete a disabled agent. Note that the agent will not be deleted unless it is in disabled state and is not building any job.

### Response Codes

HTTP response code	Explanation
200	Agent deleted successfully
404	Agent with [uuid] does not exist
406	Agent is not 'Disabled' or is still 'Building' or 'Cancelled'
401	User does not have operate permission to delete agent
500	An internal server error occurred

### A note on deleting agents

An agent continually contacts the Go server so long as it is running. If it is deleted from Go server, the next time it contacts Go server, Go will add it back to the list of agents. Normally you do not want this to happen. Follow the steps below to achieve this.

- Use the list agents API and keep checking the status of the agent that you want to delete till it shows Idle
- Use the disable agent API to disable the agent, using the UUID available from the list agent API

- Stop the Go agent service, manually or through a script. This is to prevent the agent from contacting the server again. You need to do this within 5 seconds of the disable agent API
- Use the delete agents API to delete the agent

## Examples

- We use curl, a command line tool for transferring files with URL syntax, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

The configuration with agents like this:

```
<cruise>
.....
<agents>
  <agent hostname="agent-machine1" ipaddress="10.38.10.7" uuid="e4d86ae7-71
    <resources>
      <resource>twist</resource>
      <resource>windows</resource>
    </resources>
  </agent>
  <agent hostname="agent-machine2" ipaddress="10.2.12.26" uuid="bf0e5682-51
    <resources>
      <resource>ec2</resource>
    </resources>
  </agent>
  <agent hostname="agent-machine3" ipaddress="10.2.12.26" uuid="259f7a6b-f1
  <agent hostname="agent-machine3" ipaddress="10.18.3.152" uuid="098d4904-(1
  <agent hostname="agent-machine4" ipaddress="10.18.3.153" uuid="31aea908-1
  </agents>
</cruise>
```

If you want to get all agents' information:

```
curl -u admin:badger http://goserver.com:8153/go/api/agents
```

If you want to enable agent on 'agent-machine1':

```
curl -u admin:badger -X POST http://goserver.com:8153/go/api/agents/e4d86ae7-7b7d-4b1
```

If you want to disable agent on 'agent-machine2':

```
curl -u admin:badger -X POST http://goserver.com:8153/go/api/agents/bf0e5682-51ad-418
```

If you want to delete agent on 'agent-machine4':

```
curl -u admin:badger -X POST http://goserver.com:8153/go/api/agents/31aea908-717a-438
```

## Agent Job Run History

This API lists Job instances run by an Agent in JSON format. API gives 10 instances at a time, sorted in reverse order. Supports pagination using offset which tells the API how many instances to skip. This API is built primarily to aid rendering Agent Job run history page. Hence the information required for that is exposed.

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/agents/[agent-uuid]/job_run_history/[offset]">http://[server]/go/api/agents/[agent-uuid]/job_run_history/[offset]</a>	GET	no parameters	List Agent Job Run history.

**Note:** You can get Agent's UUID from Agent listing API or Job History API.

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/> .
- We assume security has been switched on, and that there is a user named **admin** with the password **badger** .

Assuming the pipeline configuration looks like:

```
<pipelines group="first">
  <pipeline name="foo" labeltemplate="foo-${COUNT}" isLocked="true">
```

```

<materials>
  <hg url="http://10.22.12.2:8000" materialName="hg_material" />
</materials>
<stage name="DEV">
  <jobs>
    <job name="UnitTest">
      <tasks>
        <ant target="all-UAT" />
      </tasks>
    </job>
  </jobs>
</stage>
<stage name="UATTest">
  <jobs>
    <job name="UAT">
      <tasks>
        <exec command="sleep">
          <arg>10000</arg>
          <runif status="passed" />
        </exec>
      </tasks>
      <artifacts>
        <artifact src="target" dest="pkg/foo.war" />
      </artifacts>
    </job>
  </jobs>
</stage>
</pipeline>
</pipelines>

```

The following command produces output specified below:

```
curl -u admin:badger http://goserver.com:8153/go/api/agents/0794793b-5c1a-443f-b860-0
```

```
{
  "jobs": [
    {
      "agent_uuid": "0794793b-5c1a-443f-b860-df480986586b",
      "name": "UnitTest",
      "job_state_transitions": [
        {
          "state_change_time": 1411456876262,
          "id": 13,
          "state": "Scheduled"
        },
        {
          "state_change_time": 1411456881401,
          "id": 14,
          "state": "Assigned"
        },
        {

```

```
        "state_change_time": 1411456891415,
        "id": 15,
        "state": "Preparing"
    },
    {
        "state_change_time": 1411456892853,
        "id": 16,
        "state": "Building"
    },
    {
        "state_change_time": 1411456893094,
        "id": 17,
        "state": "Completing"
    },
    {
        "state_change_time": 1411456893135,
        "id": 18,
        "state": "Completed"
    }
],
"scheduled_date": 1411456876262,
"pipeline_counter": 2,
"rerun": false,
"pipeline_name": "foo",
"result": "Failed",
"state": "Completed",
"id": 3,
"stage_counter": "1",
"stage_name": "DEV"
},
{
    "agent_uuid": "0794793b-5c1a-443f-b860-df480986586b",
    "name": "UnitTest",
    "job_state_transitions": [
        {
            "state_change_time": 1411456676119,
            "id": 1,
            "state": "Scheduled"
        },
        {
            "state_change_time": 1411456757905,
            "id": 2,
            "state": "Assigned"
        },
        {
            "state_change_time": 1411456761645,
            "id": 3,
            "state": "Preparing"
        },
        {
            "state_change_time": 1411456762696,
            "id": 4,
            "state": "Building"
        },
        {
            "state_change_time": 1411456762889,
            "id": 5,
            "state": "Completing"
        },
        {
            "state_change_time": 1411456763089,
            "id": 6,
            "state": "Completed"
        }
]
```

```
{
    "state_change_time": 1411456762955,
    "id": 6,
    "state": "Completed"
}
],
"scheduled_date": 1411456676119,
"pipeline_counter": 1,
"rerun": false,
"pipeline_name": "foo",
"result": "Passed",
"state": "Completed",
"id": 1,
"stage_counter": "1",
"stage_name": "DEV"
}
],
"pagination": {
    "offset": 0,
    "total": 2,
    "page_size": 10
}
}
```

# Materials API

## Config listing API

This API allows you list all Materials in Go's Config in JSON format. This API is built primarily to aid rendering of Material modifications page. Hence only the information required for that is exposed.

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/config/materials">http://[server]/go/api/config/materials</a>	GET	no parameters	List all Material Configs.

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

Assuming the pipeline configuration looks like:

```
<pipelines group="first">
  <pipeline name="foo" labeltemplate="foo-${COUNT}" isLocked="true">
    <materials>
      <hg url="http://10.22.12.2:8000" materialName="hg_material" />
    </materials>
    <stage name="DEV">
      <jobs>
        <job name="UnitTest">
          <tasks>
            <ant target="all-UAT" />
          </tasks>
        </job>
      </jobs>
    </stage>
  </pipeline>
</pipelines>
```

The following command produces output specified below:

```
curl -u admin:badger http://goserver.com:8153/go/api/config/materials
```

```
[  
  {  
    "description": "URL: http://10.22.12.2:8000",  
    "fingerprint": "64987f67c407020dfd6badf4975421428aa5d044e0b14b3086266294b969b6a8",  
    "type": "Mercurial"  
  },  
  {  
    "description": "URL: http://10.22.12.2:7019",  
    "fingerprint": "7ee3e832cb5d0f1304502a0319f42235fb01c49fd14da0f24f0253139306ad03",  
    "type": "Git"  
  }  
]
```

## Notification API

APIs that notify Go Server when a commit has been made in Version Control and Go needs to trigger relevant pipelines.

**NOTE :** Only Go Administrators will be able to invoke this API

When using this feature, uncheck **Poll for new changes** or set **autoUpdate** flag in cruise configuration to **false** for the relevant subversion materials

## Subversion

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/material/notify/svn">http://[server]/go/api/material/notify/svn</a>	POST	uuid= [subversion repository UUID]	Notifies Go that a commit has been made to a subversion repository with the given UUID

## Examples

- We use curl, a command line tool for transferring files with URL syntax, in the

following examples. Of course, you can use any HTTP client library.

- We assume that the URL of the Go server is <http://goserver.com:8153/> .
- We assume security has been switched on, and that there is an admin user named **admin** with the password **password** .

To notify Go via SVN post commit hook

```
# File: [sub-version-repository]/hooks/post-commit
# =====
#!/bin/sh

# POST-COMMIT HOOK
#
REPOS="$1"
REV="$2"
UUID=`svnlook uuid $1` 

curl -d "uuid=$UUID" -u admin:password http://goserver.com:8153/go/api/material/notify/svn
```

## Git

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/material/notify/git">http://[server]/go/api/material/notify/git</a>	POST	repository_url= [URL of this repository configured in Go]	Notifies Go that a commit has been made to a git repository with the given URL

## Example

- We use curl, a command line tool for transferring files with URL syntax, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/> .
- We assume security has been switched on, and that there is an admin user named **admin** with the password **password** .

To notify Go via Git post commit hook

```
# File: [git-repository]/hooks/post-receive
```

```

# =====
#!/bin/sh

# POST-RECEIVE HOOK
#
echo `curl -d "repository_url=http://url-for-git-repo.git" -u admin:badger` > /dev/null
echo `curl -d "repository_url=git://url-for-git-repo.git" -u admin:badger` > /dev/null
echo `curl -d "repository_url=https://url-for-git-repo.git" -u admin:badger` > /dev/null

```

## Modifications API

This API lists Material modifications known to Go in JSON format. API gives 10 modifications at a time, sorted in reverse order. Supports pagination using offset which tells the API how many modifications to skip. This API is built primarily to aid rendering Material modifications page. Hence the information required for that is exposed.

URL format	HTTP Verb	Data
<a href="http://[server]/go/api/materials/[fingerprint]/modifications/[offset]">http://[server]/go/api/materials/[fingerprint]/modifications/[offset]</a>	GET	no parameters

**Note:** You can get Material's fingerprint from Material Config listing API.

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

Assuming the pipeline configuration looks like:

```

<pipelines group="first">
  <pipeline name="foo" labeltemplate="foo-${COUNT}" isLocked="true">
    <materials>
      <hg url="http://10.22.12.2:8000" materialName="hg_material" />
    </materials>
    <stage name="DEV">
      <jobs>
        <job name="UnitTest">
          <tasks>

```

```
        <ant target="all-UAT" />
    </tasks>
</job>
</jobs>
</stage>
</pipeline>
</pipelines>
```

The following command produces output specified below:

```
curl -u admin:badger http://goserver.com:8153/go/api/materials/7ee3e832cb5d0f1304502a
```

```
{
  "modifications": [
    {
      "modified_time": 1411463559000,
      "id": 2,
      "user_name": "Admin <test@test.com>",
      "comment": "new commit",
      "revision": "96b9157ffff67f1a3b8707d490fd72d9300f67603"
    },
    {
      "modified_time": 1409664999000,
      "id": 1,
      "user_name": "Admin <test@test.com>",
      "comment": "old commit",
      "revision": "6f75b392941c40666ae822045c064e0887ffd007"
    }
  ],
  "pagination": {
    "offset": 0,
    "total": 2,
    "page_size": 10
  }
}
```

# Configuration API

---

The Go API documented here is a work in progress. Future versions may change this API.

## Configuration versioning

---

Go stores snapshots of all valid configuration files ever used by the server. Any change to configuration through Go admin pages or filesystem, if valid, is recorded and can be retrieved anytime in future.

Go tracks these revisions through a unique fingerprint (digest) of configuration file data, and exposes an API to allow admins to retrieve any historical version, given the digest.

Digest for configuration file is reported in the response to all of the following API calls as value of HTTP header field named 'X-CRUISE-CONFIG-MD5'.

## Retrieving historical configuration snapshot

---

A user with admin privileges can invoke:

```
curl -u admin:badger http://goserver.com:8153/go/api/admin/config/[digest].xml
```

to get the version of configuration identified by *[digest]*.

For instance, if configuration as on a certain day has digest value 5059cf548db9ea2d1b9192b45529ccf0, it can be retrieved on any future day by invoking:

```
curl -u admin:badger http://goserver.com:8153/go/api/admin/config/5059cf548db9ea2d1b9192b45529ccf0.xml
```

## Other convenience APIs

In addition to 'historical configuration version retrieval', Go exposes other convenience APIs around this feature, that allow retrieval of current configuration without passing in

the digest value.

```
curl -u admin:badger http://goserver.com:8153/go/api/admin/config.xml
```

or

```
curl -u admin:badger http://goserver.com:8153/go/api/admin/config/current.xml
```

can be invoked to fetch current/latest-version of config.

## Config repository Modification listing API

This API allows you to list Config repository modifications in JSON format. This API is built primarily to aid rendering of Config repository page. Hence only the information required for that is exposed.

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/config/revisions">http://[server]/go/api/config/revisions</a>	GET	no parameters	List Config Repo modifications.

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

The following command produces output specified below:

```
curl -u admin:badger http://goserver.com:8153/go/api/config/revisions
```

```
[  
 {  
 "md5": "26fae11e99091996dc293651dd205221",
```

```

    "commitSHA": "238d2081ca5fde32440c4c1da6c6f95267196162",
    "username": "admin",
    "goEdition": "OpenSource",
    "time": 1411456839850,
    "schemaVersion": 74,
    "goVersion": "N\A"
},
{
    "md5": "24ab103b1f7c730709d0bfa188ce80c8",
    "commitSHA": "59fcc29a3385b17795c7b2ac2c2f6dd7cb9421bd",
    "username": "agent_0794793b-5c1a-443f-b860-df480986586b_192.168.1.12_unknownne4ce",
    "goEdition": "OpenSource",
    "time": 1411456761631,
    "schemaVersion": 74,
    "goVersion": "N\A"
}
]

```

## Config repository Modification Diff API

This API allows you to get diff between 2 modifications in Config Repo in JSON format.

This API is built primarily to aid rendering of Config repository page. Hence only the information required for that is exposed.

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/config/diff/[from-SHA]/[to-SHA]">http://[server]/go/api/config/diff/[from-SHA]/[to-SHA]</a>	GET	no parameters	Diff between 2 Config Repo modifications.

## Examples

- We use curl, a command line tool to demonstrate the use of the API, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

The following command produces output specified below:

```
curl -u admin:badger http://goserver.com:8153/go/config/diff/238d2081ca5fde32440c4c1c
```

```

@@ -55,7 +55,7 @@
  </pipelines>
  <agents>
    <agent hostname="blrstdcrspair02" ipaddress="10.4.7.202" uuid="f6a76eaa-96ac-43a
-
-    <agent hostname="unknowne4ce8f40e2ca" ipaddress="127.0.0.1" uuid="0794793b-5c1a-
+    <agent hostname="unknowne4ce8f40e2ca" ipaddress="192.168.1.12" uuid="0794793b-5c
  </agents>
</cruise>

```

## Adding a new pipeline

To add a pipeline, you perform a POST to the URL

[http://\[server\]/go/tab/admin/pipelines/\[pipeline\\\_name\].json](http://[server]/go/tab/admin/pipelines/[pipeline\_name].json) where pipeline\_name is the name of the pipeline that you wish to create. Creating a pipeline supports the following parameters:

For example, suppose you have switched on security and the username and the password are 'my\_user' and 'my\_password'. If you want to create a new pipeline named 'mypipeline', which uses an svn repository without username and password, and the location of repository is '<http://yoursvnrepository/trunk>'. The command should be:

```
curl -u admin:badger -d "url=http://yoursvnrepository/trunk" http://goserver.com:8151/go/tab/admin/pipelines/mypipeline.json
```

## General parameters

parameter name	example value	required	Description
pipelineGroup	defaultGroup	No	The name of the Pipeline Group to add the new pipeline to. The pipeline group will be created if it does not already exist.
builder	ant	No	Can be 'ant', 'nant', 'rake' or 'exec'.
buildfile	build.xml	No	Not allowed for exec
target	all	No	Not allowed for exec
command	unittest.sh arg1 arg2	No	Required for exec

source	pkg	No	no default value
dest	installer	No	no default value

## Subversion

parameter name	example value	required	Description
scm	svn	No	Default value is 'svn'
url	<a href="http://xxx.xx.xx.xx/mysvn/trunk">http://xxx.xx.xx.xx/mysvn/trunk</a>	Yes	The URL of the Subversion repository
username	checkout_username	No	
password	checkout_password	No	

## Mercurial

parameter name	example value	required	Description
scm	hg	Yes	
url	<a href="http://xxx.xx.xx.xx/hg/my_project">http://xxx.xx.xx.xx/hg/my_project</a>	Yes	The URL of the repository

## Git

parameter name	example value	required	Description
scm	git	Yes	
url	<a href="git://github.com/foo/bar.git">git://github.com/foo/bar.git</a>	Yes	The URL of the repository

## Perforce

parameter name	example value	required	Description
scm	p4	Yes	
url	p4server.foo.com:1666	Yes	The P4PORT of the repository

username	checkout_username	No	The P4USER to connect to the repository
password	checkout_password	No	The P4PASSWD to connect to the repository
useTickets	true or false	No	
view	//depot/... //something/...	Yes	

## Team Foundation Server

parameter name	example value	required	Descr
scm	tfs	Yes	
url	<a href="http://tfs.host.com:8080/tfs/DefaultCollection">http://tfs.host.com:8080/tfs/DefaultCollection</a>	Yes	The url to your TFS collection
domain	COMPANYNAME	No	Domain name of the given user belonging to
username	tfssuser	Yes	User name used to connect to the collection
password	tfspassword	No	Password for the user name
projectPath	\$/MyProject	Yes	Project path in the collection

# Artifacts API

The Go API documented here is a work in progress. Future versions may change this API.

The Artifacts API is not as much an API as a way to list, download and upload artifacts through command-line.

## List

Method	URL format	HTTP Verb	Explanation
List	<code>http://[server]/go/files/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job].json</code>	GET	List all files for the particular pipeline/stage/job in json format

## Show

Method	URL format	HTTP Verb	Explanation
Show	<code>http://[server]/go/files/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job]/[filename]</code>	GET	Get the file called [filename] in default artifact folder of the pipeline/stage/job with particular label.
Show	<code>http://[server]/go/files/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job]/[pathname]/[filename]</code>	GET	Get the file called [filename] in the sub-folder [pathname] of default artifact folder of the pipeline/stage/job with particular pipeline counter.
Show	<code>http://[server]/go/files/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job]/[pathname.zip]</code>	GET	Get a zipped output of all the files in the [pathname] of default artifact folder of the pipeline/stage/job with particular

# Create & Append

Method	URL format	HTTP Verb	Explanation
Create	<code>http://[server]/go/files/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job]/[filename]</code>	POST	upload a file named [filename] to the default artifact folder of the particular pipeline/stage/job
Append	<code>http://[server]/go/files/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job]/[filename]</code>	PUT	appending a string to a file named 'filename' in the default artifact folder of the pipeline/stage/job with particular pipeline counter

- You can use key word 'latest' as a pipeline counter or a stage counter.
- **pipeline-counter** is a number which indicates how many times the pipeline has been triggered.
- **stage-counter** is a number which indicates how many times the stage has been re-run in the pipeline with the same pipeline counter.
- The name in the RESTful url is case-sensitive.

## Examples

- We use curl, a command line tool for transferring files with URL syntax, in the following examples. Of course, you can use any HTTP client library.
- We assume that the url of the Go server is `http://goserver.com:8153/`.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger**.

And the pipeline configuration looks like:

```
<pipeline name="foo" labeltemplate="foo-1.0-${COUNT}">
  <materials>
    <svn url="...."/>
  </materials>
  <stage name="DEV">
```

```

<jobs>
  <job name="UnitTest">
    <tasks>
      <ant target="ut"/>
    </tasks>
    <artifacts>
      <artifact src="coverage" dest="coveragereport.html"/>
    </artifacts>
  </job>
</jobs>
</stage>
<stage name="UATest">
  <jobs>
    <job name="UAT">
      <tasks>
        <ant target="all-UAT"/>
      </tasks>
      <artifacts>
        <artifact src="report" dest="UAreport.html"/>
        <artifact src="target" dest="pkg/foo.war"/>
      </artifacts>
    </job>
  </jobs>
</stage>
</pipeline>

```

If you want to get the list of files in json for the latest completed job UnitTest, the command is

```
curl -u admin:badger http://goserver.com:8153/go/files/foo/latest/DEV/1/UnitTest.json
```

If you want to get the list of files in json for the job UnitTest with the pipeline counter 1243 and stage counter '1', the command is

```
curl -u admin:badger http://goserver.com:8153/go/files/foo/1243/DEV/1/UnitTest.json
```

If you want to get the file 'foo.war' under the folder 'pkg' for the job UAT with the pipeline counter 1243, the command is

```
curl -u admin:badger http://goserver.com:8153/go/files/foo/1243/UATest/1/UAT/pkg/foo.war
```

If you want to get a zip file of the contents of 'binaries' under the folder 'pkg' for the job

UAT with the pipeline counter 1243, the command is

```
curl -u admin:badger http://goserver.com:8153/go/files/foo/1243/UATest/1/UAT/pkg/binar
```

The above command will return immediately with an HTTP status code of 202 to indicate that the request was accepted. The same command needs to be retried after a few seconds to actually retrieve the zip contents

If you want to get the list in the folder "pkg" in json format for the job UAT with the pipeline counter 1243 with stage counter '1', the command is

```
curl -u admin:badger http://goserver.com:8153/go/files/foo/1243/UATest/1/UAT/pkg.json
```

If you want to upload a file abc.txt to the job UAT with the pipeline counter 1243 and stage counter '1', and the target name is def.txt, the command is

```
curl -u admin:badger -F file=@abc.txt http://goserver.com:8153/go/files/foo/1243/UATest/1/UAT/pkg/def.txt
```

If you want to upload a zip file abc.zip to the job UAT with the pipeline counter 1243 with stage counter '1', and the target name is def.zip, the command is

```
curl -u admin:badger -F file=@abc.zip http://goserver.com:8153/go/files/foo/1243/UATest/1/UAT/pkg/def.zip
```

### Notes:

In order to upload a folder, first zip it up. Next, assuming you want it to be associated with the job UAT, pipeline counter 1243, stage counter '1', and the folder name 'def', the command is

```
curl -u admin:badger -F zipfile=@abc.zip http://goserver.com:8153/go/files/foo/1243/UATest/1/UAT/pkg/def
```

If you want to append the content of a file, say abc.txt, to the end of a file 'def.txt' to the job UnitTest with the pipeline counter 1243 and stage counter '1', the command is

```
curl -u admin:badger -T abc.txt http://goserver.com:8153/go/files/foo/1243/DEV/1/Unit
```

# Users API

---

This API allows you to control User accounts in Go

**NOTE:** Only Go administrator users will be able to use this API

## Delete

---

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/users/[user_name]">http://[server]/go/api/users/[user_name]</a>	DELETE	no parameters	Deletes the user with username 'user_name'. Note: Only disabled users can be deleted.

### Return codes

HTTP Code	Reason
200	User deleted successfully
400	The user to be deleted is not disabled.
401	User executing the request is unauthorized to perform this operation.
404	The user to be deleted does not exist.

## Examples

---

- We use curl, a command line tool for transferring files with URL syntax, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is an admin user named **admin** with the password **badger**.

To delete a user named 'another\_user'

---

```
curl -u admin:badger -X "DELETE" http://goserver.com:8153/go/api/users/another_user
```

# Backup API

---

This API allows you to backup Go server. Please refer to [Backup Go Server](#) to get more information about the backup functionality.

**NOTE:** Only Go administrator users will be able to use this API

## Trigger Backup

---

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/admin/start_backup">http://[server]/go/api/admin/start_backup</a>	POST	no parameters	Go server backup initiates

## Examples

---

- We use curl, a command line tool for transferring files with URL syntax, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is an admin user named **admin** with the password **badger**.

To initiate the backup

```
curl -u admin:badger -X POST http://goserver.com:8153/go/api/admin/start_backup
```

# Properties API

The Go API documented here is a work in progress. Future versions may change this API.

You can get the list of properties or a property's value for a given job by using the properties API.

There is no way to delete or update a property.

StageCounter is a number which indicate how many times the stage has been run in the pipeline with the same pipeline label.

## List & Show

Method	URL format	HTTPVerb	Explanation
List	<a href="http://[server]/go/properties/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job]">http://[server]/go/properties/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job]</a>	GET	List all properties for the pipeline with counter in CSV format.
Show	<a href="http://[server]/go/properties/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job]/[property-name]">http://[server]/go/properties/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job]/[property-name]</a>	GET	Get the property value for the pipeline with specific counter name.

## Create

URL format	HTTPVerb	Explanation
<a href="http://[server]:8153/go/properties/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job]/[property-name]">http://[server]:8153/go/properties/[pipeline]/[pipeline-counter]/[stage]/[stage-counter]/[job]/[property-name]</a>	POST	Create a property with value to specific pipeline/stage/counter.

## Search

URL format	HTTPVerb	Explanation

<pre>"http://[server]:8153/go/properties/search? pipelineName=[pipeline]&amp;stageName= [stage]&amp;jobName=[job]&amp;limitPipeline= [pipeline-counter]&amp;limitCount=[number]"</pre>	GET	<p>List all historical properties for the pipeline/stage/job upto specified pipeline in csv format. The limitPipeline is optional, which is the last pipeline counter in the list and the default value is the latest pipeline instance. The limitCount is the number of pipeline instances that Go should return. ;limitCount is optional and its default value is 100.</p>
--	-----	--

- You can use key word 'latest' as a pipeline counter or a stage counter.
- RESTful urls are case sensitive.
- Go does not support JSON format for properties API.

## Examples

---

- We use curl, a command line tool for transferring files with URL syntax, in the following examples. Of course, you can use any HTTP client library.
- We assume that the url of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is a user named **admin** with the password **badger** .

And the pipeline configuration looks like:

```
<pipeline name="foo" labeltemplate="foo-1.0-${COUNT}">
  <material>
    <svn url="...."/>
  </material>
  <stage name="DEV">
    <job name="UnitTest">
      <tasks>
        <ant target="ut"/>
      </tasks>
      <artifacts>
        <artifact src="coverage" dest="coveragereport.html"/>
      </artifacts>
    </job>
  </stage>
  <stage name="UATest">
    <job name="UAT">
```

```
<tasks>
    <ant target="all-UAT"/>
</tasks>
<artifacts>
    <artifact src="report" dest="UAreport.html"/>
    <artifact src="target" dest="pkg/foo.war"/>
</artifacts>
</job>
</stage>
</pipeline>
```

If you want to get the list of properties in csv for the job UnitTest with the pipeline counter '1243' and stage counter 'LATEST', the command is

```
curl -u admin:badger http://goserver.com:8153/go/properties/foo/1243/DEV/LATEST/Unit:
```

If you want to get the history of properties in csv for the job UnitTest, the command is

```
curl -u admin:badger "http://goserver.com:8153/go/properties/search?pipelineName=foo&
```

The quotes are required.

If you want to get a property 'cruise\_agent' for the job UnitTest with the pipeline counter 1243 and stage counter '1', the command is

```
curl -u admin:badger http://goserver.com:8153/go/properties/foo/1243/DEV/1/UnitTest/c
```

If you want to define a property, named myproperty, for the job UnitTest with the pipeline counter '1243' and stage counter '1', the command is

```
curl -u admin:badger -d "value>Showcase_for_I29" http://goserver.com:8153/go/propert:
```

# Feeds API

The Go API documented here is a work in progress. Future versions may change this API.

Unless specified otherwise, all the api requests enforce user security. For example, if security is turned on, and a user has access to only two pipelines, all api requests will return data for those two pipelines.

## List

URL format	HTTP Verb	Explanation
<a href="http://[server]/go/api/pipelines.xml">http://[server]/go/api/pipelines.xml</a>	GET	List of all pipelines
<a href="http://[server]/go/api/pipelines/[pipeline]/[pipeline-id].xml">http://[server]/go/api/pipelines/[pipeline]/[pipeline-id].xml</a>	GET	Feed of a pipeline
<a href="http://[server]/go/api/pipelines/[pipeline]/stages.xml">http://[server]/go/api/pipelines/[pipeline]/stages.xml</a>	GET	Feed of all stages for the pipeline [pipeline_name]
<a href="http://[server]/go/api/stages/[stage-id].xml">http://[server]/go/api/stages/[stage-id].xml</a>	GET	Xml representation of a stage
<a href="http://[server]/go/pipelines/[pipeline]/[pipeline-counter]/[stage]/[stage-counter].xml">http://[server]/go/pipelines/[pipeline]/[pipeline-counter]/[stage]/[stage-counter].xml</a>	GET	Xml representation of a stage
<a href="http://[server]/go/pipelines/[pipeline]/[pipeline-label]/[stage]/[stage-counter].xml">http://[server]/go/pipelines/[pipeline]/[pipeline-label]/[stage]/[stage-counter].xml</a>	GET	Xml representation of a stage
<a href="http://[server]/go/api/jobs/[job-id].xml">http://[server]/go/api/jobs/[job-id].xml</a>	GET	Xml representation of a job

- [pipeline-id] is a unique identifier for a pipeline run.
- [stage-id] is a unique identifier for a stage run.
- [job-id] is a unique identifier for a stage run.
- The urls are case-sensitive.

## Examples

- We use curl, a command line tool for transferring files over HTTP, in the following examples. Of course, you can use any HTTP client library.
- We assume that the url of the Go server is <http://goserver.com:8153> .
- We assume security has been switched on, and that there is a user named **admin** with the password **badger** .

Given a pipeline configuration like:

```
<pipeline name="go_pipeline" labeltemplate="go-1.0-${COUNT}">
  <materials>
    <svn url="...."/>
  </materials>
  <stage name="DEV">
    <jobs>
      <job name="UnitTest">
        <tasks>
          <ant target="ut"/>
        </tasks>
      </job>
    </jobs>
  </stage>
  <stage name="UATest">
    <jobs>
      <job name="UAT">
        <tasks>
          <ant target="all-UAT"/>
        </tasks>
      </job>
    </jobs>
  </stage>
</pipeline>
```

You can get a list of all pipelines using the following command:

```
curl -u admin:badger http://goserver.com:8153/go/api/pipelines.xml
```

This would return xml that looks similar to:

```
<pipelines>
  <link rel="self" href="http://goserver.com:8153/go/api/pipelines.xml"/>
  <pipeline href="http://goserver.com:8153/go/api/pipelines/go_pipeline/stages.xml">
</pipelines>
```

You can now get a feed of completed stages for a pipeline. In the feed entry you can get

information about the completed stage, pipeline it belongs to, names of the people who checked in to trigger a given pipeline and links to Mingle cards that were worked upon.

Currently stages.xml does not provide results when an upstream pipeline is renamed.

The following is the command to get the feed for a pipeline called 'go\_pipeline':

```
curl -u admin:badger http://goserver.com:8153/go/api/pipelines/go_pipeline/stages.xml
```

This would return an ATOM feed that looks similar to the following:

```

<id>http://goserver.com:8153/go/pipelines/go_pipeline/268/build/1</id>

<author>
  <name>Dev2 <another _ dev@organization.com></name>
</author>

<link title="build Stage Detail" href="http://goserver.com:8153/go/api/stages/268/build/1/stages/76"/>
<link title="build Stage Detail" href="http://goserver.com:8153/go/pipelines/268/build/1/stages/76"/>
<link title="go_pipeline Pipeline Detail" href="http://goserver.com:8153/go/api/pipelines/268"/>
<link title="go_pipeline Pipeline Detail" href="http://goserver.com:8153/go/pipelines/268"/>

<link href="https://mingleserver.com/api/v2/projects/some_project/cards/2408"/>
<link href="https://mingleserver.com/projects/some_project/cards/2408" rel="self"/>

<category scheme="http://www.thoughtworks-studios.com/ns/categories/go" term="stage"/>
<category scheme="http://www.thoughtworks-studios.com/ns/categories/go" term="build"/>
<category scheme="http://www.thoughtworks-studios.com/ns/categories/go" term="pipeline"/>
</entry>
</feed>

```

All tags will have a CDATA element wherever appropriate in order to make the XML valid

If you want details of a particular stage, use the alternate link provided in each entry:

```
curl -u admin:badger http://goserver.com:8153/go/api/stages/76.xml
```

**Note:** stage-id in the url above is 76, which matches the first feed entry above.

Additionally, you can directly get the same xml if you know the pipeline name and stage name

For example, in the case above, you could use the following command

```
curl -u admin:badger http://goserver.com:8153/go/pipelines/go_pipeline/2/DEV/1.xml
```

Notice that this is the url one gets by appending .xml to the id element for the stage entry above

The returned xml looks similar to:

```
<stage name="dev" counter="1">
```

```

<link rel="self" href="http://goserver.com:8153/go/api/stages/65615.xml"/>
<id>urn:x-go.studios.thoughtworks.com:stage-id:go_pipeline:2:dev:1</id>
<pipeline name="go_pipeline" counter="2" label="2.1.2322" href="http://goserver.com:8153/go/api/pipelines/2.xml"/>
<updated>2010-11-02T17:49:32+05:30</updated>
<result>Passed</result>
<state>Completed</state>
<approvedBy>changes</approvedBy>
<jobs>
  <job href="http://goserver.com:8153/go/api/jobs/45.xml" />
  <job href="http://goserver.com:8153/go/api/jobs/46.xml" />
</jobs>
</stage>

```

You can now get the details of each job in this stage by using the urls in the job elements. For example:

```
curl -u admin:badger http://goserver.com:8153/go/api/jobs/45.xml
```

The returned xml looks similar to:

```

<job name="build">
  <link rel="self" href="http://goserver.com:8153/go/api/jobs/45.xml"/>
  <id>urn:x-go.studios.thoughtworks.com:job-id:acceptance:1039:twist:1:firefox-7</id>
  <pipeline name="go_pipeline" counter="2" label="2.1.2322" href="http://goserver.com:8153/go/api/pipelines/2.xml"/>
  <stage name="DEV" counter="1" href="http://goserver.com:8153/go/api/stages/76.xml"/>
  <state>Completed</state>
  <result>Failed</result>
  <properties>
    <property name="cruise_agent">dev-agent</property>
    <property name="cruise_job_duration">1906</property>
    <property name="cruise_job_id">10</property>
    <property name="cruise_job_result">Passed</property>
    <property name="cruise_pipeline_counter">2</property>
    <property name="cruise_pipeline_label">2.1.2322</property>
    <property name="cruise_stage_counter">1</property>
    <property name="cruise_timestamp_01_scheduled">2010-09-22T12:36:15+05:30</property>
    <property name="cruise_timestamp_02_assigned">2010-09-22T12:36:24+05:30</property>
    <property name="cruise_timestamp_03_preparing">2010-09-22T12:36:34+05:30</property>
    <property name="cruise_timestamp_04_building">2010-09-22T12:36:39+05:30</property>
    <property name="cruise_timestamp_05_completing">2010-09-22T13:08:26+05:30</property>
    <property name="cruise_timestamp_06_completed">2010-09-22T13:08:26+05:30</property>
    <property name="tests_failed_count">2</property>
    <property name="tests_ignored_count">0</property>
    <property name="tests_total_count">15</property>
    <property name="tests_total_duration">1599.552</property>
  </properties>
  <agent uuid="2e9c36c1-5a41-43ad-85a7-8195f179388c"/>
  <artifacts baseUri="http://goserver.com:8153/go/files/go_pipeline/2/DEV/1/build">
    <artifact path="cruise-output/log.xml" type="file"/>
    <artifact path="server/logs/*.log" type="file"/>
  </artifacts>
</job>

```

```
<artifact path="server/config/config" type="file"/>
<artifact path="server/pipelines/pipelines" type="file"/>
<artifact path="xml" type="unit"/>
</artifacts>
<resources>
    <resource>dev</resource>
    <resource>smoke</resource>
</resources>
<environmentvariables>
    <variable name="COMPRESS_JS">
        <value>No</value>
    </variable>
    <variable name="COMPRESS_CSS">
        <value>Yes</value>
    </variable>
</environmentvariables>
</job>
```

# Command Repo API

---

This API allows you to reload the command repository cache on the Go server. Please refer to [Command Repository](#) to get more information about the functionality.

**NOTE:** Only Go administrator users will be able to use this API

## Reload

---

URL format	HTTP Verb	Data	Explanation
<a href="http://[server]/go/api/admin/command-repo-cache/reload">http://[server]/go/api/admin/command-repo-cache/reload</a>	POST	no parameters	Go server reloads command repository cache

## Examples

---

- We use curl, a command line tool for transferring files with URL syntax, in the following examples. Of course, you can use any HTTP client library.
- We assume that the URL of the Go server is <http://goserver.com:8153/>.
- We assume security has been switched on, and that there is an admin user named **admin** with the password **badger**.

To reload the cache

```
curl -u admin:badger -X POST http://goserver.com:8153/go/api/admin/command-repo-cache/reload
```

# Go Plugin User Guide

---

Please go through [Go Plugin User Guide](#) before using Go plugin

## Available Go Extension Points

---

- Package Repository Extension
- Task Extension

## Package Repository Extension

---

Please follow [link](#) for package material extension point details.

- [Yum Repository Poller](#)

## Task Extension

---

Please follow [link](#) for task extension point details. Go does not have

## Note

---

Please refer [Developer Documentation] (<http://www.go.cd/documentation/developer/#8-writing-go-plugins>) if you are planning to write plugin

# Plugin User Guide

---

## Introduction

---

Plugins allow users to extend the functionality of Go. Each plugin is assigned an identifier which is determined by the **id** attribute provided in plugin metadata file packaged along with the plugin jar. If the metadata file is not packaged, plugin jar file name will be taken as plugin id. Plugins are classified into two categories - Bundled and External. During startup, Go server would try to load all the plugins. On successful load, the plugin will be converted into an OSGi bundle and extracted into **< server installation directory >/plugins\_work** directory. **Plugins** tab, under Go server Administration, would list all the loaded plugins.

## Bundled versus External

---

- **Bundled Plugins:** As the name suggests, bundled plugins are bundled/packaged along with Go server. These are developed and supported by Thoughtworks Go development team. Bundled plugins are located under **< server installation directory >/plugins/bundled** directory. After an upgrade, when Go server starts up for the first time, all bundled plugins would be refreshed with the latest packaged versions.
- **External Plugins:** All user authored plugins are treated as external plugins. Unlike bundled plugins, external plugins are not bundled/refreshed/removed/modified during a Go server upgrade. External plugins are loaded from **< server installation directory >/plugins/external** directory. [Listing of external plugins](#)

## Installing and Uninstalling of plugins

---

You need access to the Go server machine to be able to install/uninstall a plugin. To install a plugin, drop the plugin jar under the external plugin directory ( **< server installation directory >/plugins/external** ) and restart Go server. To uninstall a plugin, remove the plugin jar from the external plugin directory ( **< server installation directory >/plugins/external** ) and restart Go server.

# Plugins Tab

Plugins tab can be found under Go server Administration. Plugins tab shows all the plugins that have been loaded currently along with its details and status. If a plugin is marked invalid or incompatible, the reasons for the same would be reported here.

Administration

Pipelines   Templates   Config XML   Server Configuration   User Summary   OAuth Clients   OAuth Enabled Gadget Providers   Backup   Plugins   Package Repositories

Plugins

---

 **Yum plugin** 1.0.0 [yum]  
Plugin that polls a yum repository  
[More Information](#)

Author ThoughtWorks Go Team

Loaded from: /var/lib/go-server/plugins/bundled/yum-repo-exec-poller.jar  
Target operating systems: Linux  
Target Go Version: 13.3  
Bundled: Yes

---

## Notes

- Add/delete/upgrade of a plugin will take effect only after a Go server restart.
- Two plugins cannot have same **id** irrespective of whether it is a bundled or an external plugin.
- If two external plugins with same **id** are available, only one of them will be loaded successfully in no specific order.
- If a bundled and an external plugin with same **id** are available, only bundled plugin will be loaded.

# Package Material

---

## Introduction

---

Pipelines in Go can poll packages in repositories similar to how they poll version control systems. A build typically consumes source code maintained in a version control system (VCS/SCM). What about a typical deployment? Increasingly, the input for deployments is the build result packaged as:

1. jar, war or ear file in case of Java
2. [nuget/ chocolatey](#) package in case of .NET
3. [Linux system package](#) (e.g rpm, deb) in case of any application platform
4. Other [application level package formats](#) like gem, npm, phar, wheel etc.

These files (packages) are often maintained in corresponding package repositories.

**Such packages may be specified as materials for Go pipelines.**

## Supported Packages

Since there are many package formats each with its own package manager and repository, the support for package-as-material has been implemented as an extension point. Using the bundled [yum-repo-poller plugin](#), it is possible to specify an rpm package held in a yum repository as a material for a Go pipeline. Using other [external plugins](#), it is possible to do the same for other types of packages.

## Repositories, Packages and Materials

A repository may contain one or more packages. A pipeline may refer to a package as a material. When the package is updated in the repository, interested pipelines will get scheduled.

## Repository Definition

A package material plugin lets pipeline group admins provide details of the corresponding repository type to Go. e.g. here is how we define a yum repository using the bundled [yum-repo-poller plugin](#).

**Note:**

1. The repository name is not used by the package material plugin - it is used by Go to construct the material name.
2. Two package repositories cannot have the same name.
3. Use the check connection button to ensure that Go can work with this repository.

**Edit Package Repository**

Name \*  Enter a globally unique name

Type \*  Choose from available plugins

YUM Repository Configuration

Repository URL \*

User

Password

Change Password

**CHECK CONNECTION**

\* indicates a required field

**SAVE** **RESET**

## Package Definition

A package material plugin also lets you define packages at the time of pipeline material configuration (Admin > Material >Add Material > Package). Here is what it looks like for defining RPM packages with the bundled yum plugin. The package name is not used by the package material plugin - it is used by Go to construct the material name. Two packages in a repository cannot have the same name. Use the check package button to ensure that the package definition does indeed resolve to the package you are looking for.

**Edit Material - Package**

Repository\*  Choose a configured repository

Package\*  Choose Existing  Define New

---

— New Package —

*i* The new package will be available to be used as material in all pipelines. Other admins might be able to edit this package.

Package Name\*

Package Spec\*

**CHECK PACKAGE** OK. Found package 'gcc-4.4.7-3.el6.x86\_64'.

\* indicates a required field

**SAVE PACKAGE AND MATERIAL** **CANCEL**

Unlike other VCS/SCM materials, *the material definition in case of packages is not contained within the pipeline definition*. Many pipelines may have material definitions referring to the same package. Here is how we associate an existing package as material for a pipeline.

**Edit Material - Package**

Repository\*  Choose a configured repository

Package\*  Choose Existing  Define New

git  
[Select]  
git  
ant

---

— Package Configuration —

Package Name

Package Spec

**CHECK PACKAGE**

\* indicates a required field

**SAVE** **CANCEL**

**Note:**

Each package definition must resolve to exactly one package on the repository, else the pipeline will not run. In order to set up a pipeline that polls for multiple packages, configure each package as a separate material.

Each package material plugin defines a subset of its properties as a *package fingerprint*. e.g. in case of the bundled yum plugin this subset consists of Repository URL and Package Spec (it excludes repository username and password). Repository and Package names are **not** part of package fingerprint. It is not permitted to multiple packages having the same package fingerprint. An attempt to do so will result in an error message like this:

The following error(s) need to be resolved in order to perform this action:  
Cannot save package or repo, found duplicate packages. [Repo Name: 'orchard',  
Package Name: 'apple'], [Repo Name: 'orchard', Package Name: 'orange']

## Sample XML Configuration

Here is a XML view of an RPM package defintion. Note the relation between repository, package and pipeline material. Loosely typed property, key and value tags are used for repository and package configuration in order to accommodate different plugins. If you choose to configure via direct XML edit, note that it isn't necessary to provide repository and package IDs, Go server wil autogenerate them. However, not all validations that are performed while configuring via UI kick in while configuring via XML edit - the resulting failures will show up later in the server health message panel at the bottom right of the browser frame.

```
< repository id="1ce5c205-977f-4c0e-ada4-882030580eed" name="ora" >
  <pluginConfiguration id="yum" version="1" />
  <configuration>
    <property>
      <key>REPO_URL</key>
      <value>http://public-yum.oracle.com/repo/OracleLinux/OL6/latest/x86_64</value>
    </property>
  </configuration>
  <packages>
    <package id="840b0b60-bd29-489d-b5ea-ccff5f6459a9" name="gcc" autoUpdate="false">
      <configuration>
        <property>
          <key>PACKAGE_SPEC</key>
          <value>gcc-4.*</value>
        </property>
      </configuration>
    </package>
  </packages>
</repository>
...
```

```
< pipelines group="showcase">
  <pipeline name="dependsOnGcc">
    <materials>
      <package ref="840b0b60-bd29-489d-b5ea-ccff5f6459a9" />
    </materials>
  ...

```

## Value stream modeling tip

Depending on whether Go is also publishing the package or just consuming it, there are two options for modeling a value stream that includes packages.

1. The first scenario is where the package is published from some pipeline in Go. Say pipeline X publishes package P to an external repo and pipeline Y consumes P. To trigger Y after publication of P, there are two options:
  - i. Pipeline dependency: X becomes a material for Y. Y resolves the exact version of P and downloads it on its own (although this [tip](#) may be used to pass package version information from X to Y). X will appear as an upstream component of Y in the [value stream map](#).
  - ii. Package material: Y adds P as a package material. Y no longer has to resolve P.

It isn't advisable to do both as Y will then schedule twice. The choice depends on how closely the activities in pipeline X and Y are related. If it is important to see X and Y together in the same value stream map, then option #1 makes sense.

2. The second scenario is where Go does not know about how/who published the package. Perhaps it got published by a job in Jenkins. Or perhaps the package is an open source package on a public repository on the internet. In this case the only option is to use a package material. Go helps you trace back to the external origin of the package if the package creator adds trackback information to the package metadata. The details of this will vary by plugin. In case of the bundled yum plugin, we use the URL field in rpm metadata for this.

## Permissions

Repositories and their packages are global entities not tied to a pipeline group or environment. Pipeline group admins may define repositories and packages for use in their pipelines. One pipeline group admin may also use packages defined by another for their pipelines. Changing a pacakge definition will cause all dependent pipelines to schedule - even those not in the same pipeline group as that of the person editing.

Because of this, we don't have a UI way of changing the definition of a package. Only the Go admin can change it via Admin > Config XML tab.

## Polling

Even if no pipelines use a package, Go polls for newer packages every minute. This may be turned off at a package level by setting `autoUpdate` to false via the config xml (Go admins only). `autoUpdate` is turned on by default. When a newer package is found, the pipelines for which it is a material get scheduled (assuming [auto scheduling of pipelines](#) is on). Also see [API scheduling](#).

## Package information display

The following information is expected from the package material plugin (which in turn obtains it from the package metadata if available)

1. Package revision (e.g. gcc-4.4.7-3.el6.x86\_64)
2. Package build time
3. Name of package creator (if available)
4. Package comment
5. Trackback URL - Typically an absolute URL that indicates what job (in Go or otherwise) created the package.
6. Package - material name (i.e. repo-name:package-name)

At the time of building the package, it is recommended to include as much of the above information as possible so that it is available for Go to display as below.

The screenshot shows the Go UI interface for a pipeline named 'p1'. The top navigation bar includes 'Label: 4', 'Compare', and 'Changes' dropdown. The main content area displays three package builds:

Build	Creator	Build Time	Built On	Trackback	Material Name
1	anonymous	2013-09-04T11:28:44+05:30	go-qqa3	Not Provided	go-agent-13.3.0-17938.noarch
2	anonymous	2012-02-09T01:04:23+05:30	ca-build44.us.oracle.com	http://www.oracle.com/virtualization	libovmapi-3.0-6.el6.x86_64
3	anonymous	2013-05-29T06:12:45+05:30	ca-build44.us.oracle.com	Not Provided	oracle-em-agent-12cR1-preinstall-1.0-4.el6.x86_64

At the bottom left, there are navigation arrows and a copyright notice: 'Copyright © 2013'.

## Downloading the package

The package isn't automatically downloaded on the agent and made available to the jobs. This is unlike VCS/SCM materials where a checkout is made by default. In case of packages, the Go Agent is typically not the target node for deployment, it is only orchestrating deployment to a remote node. So, instead of an automatic download, the following environment variables are made available:

1. GO\_PACKAGE\_< REPO-NAME >\_< PACKAGE-NAME >\_LABEL
2. GO\_REPO\_< REPO-NAME >\_< PACKAGE-NAME >\_REPO\_URL
3. GO\_PACKAGE\_< REPO-NAME >\_< PACKAGE-NAME >\_LOCATION

Repository and package names are converted to all uppercase and hyphens are converted to underscores before inclusion in the environment variable names. For example, let's say we set up a repository named ORA pointing to [http://public-yum.oracle.com/repo/OracleLinux/OL6/latest/x86\\_64](http://public-yum.oracle.com/repo/OracleLinux/OL6/latest/x86_64) and define a package gcc with a spec of gcc-4.\* and set it up as material for a pipeline. To download the package locally on the agent, we could write a task like this:

```
[go] Start to execute task: <exec command="/bin/bash" >
<arg>-c</arg>
<arg>curl -o /tmp/gcc.rpm $GO_PACKAGE_ORA_GCC_LOCATION</arg>
</exec>
```

When the task executes on the agent, the environment variables get substituted as below:

```
[go] Start to execute task: <exec command="/bin/bash" >
<arg>-c</arg>
<arg>curl -o /tmp/gcc.rpm $GO_PACKAGE_ORA_GCC_LOCATION</arg>
</exec>.
...
[go] setting environment variable 'GO_PACKAGE_ORA_GCC_LABEL' to value 'gcc'
[go] setting environment variable 'GO_REPO_ORA_GCC_REPO_URL' to value 'http://public-yum.oracle.com/repo/OracleLinux/OL6/latest/x86_64'
[go] setting environment variable 'GO_PACKAGE_ORA_GCC_PACKAGE_SPEC' to value 'gcc-4.*'
[go] setting environment variable 'GO_PACKAGE_ORA_GCC_LOCATION' to value '/tmp/gcc.rpm'
...
```

Or, to simply pass it as an argument to a deploy script on a remote server

```
<exec command="/bin/bash">
<arg>-c</arg>
<arg>ssh server "cd /to/dest/dir;deploy.sh $GO_PACKAGE_ORA_GCC_LOCATION"</arg>
</exec>
```

## Also see [Installing RPMs](#)

**Important:** Please note that if you change repository credentials and then try to re-trigger (redeploy) an old package, the published environment variables will not reflect new credentials.

## Publishing a Package

At the moment, Go does not create or publish the package for you. But it is simple enough for each type of package. e.g. [rpm](#)

You could also explore the command repository on [GitHub](#) for helpful commands. What is command repository? Please see [this](#).

## Package Dependencies

Please note that Go does not support any sort of automatic polling or other support for package dependencies. Each pacakge dependency has to specified as a separate material if needed. Alternatively, just poll for the packages at the root of the dependency graph and let the package manager figure out the rest at the time of installation. e.g. if componentA-1.2.0-b234-noarch.rpm depends on componentB-2.3.0 or above, simply poll for componentA and let

```
yum install componentA-1.2.0-b234-noarch
```

do the resolution of componentB for you.

# Yum Repository Poller

**Note:** This plugin is available for Go servers running on Linux nodes having repoquery installed (part of the [yum-utils](#) package. [Ubuntu](#), [CentOS](#))

# Introduction

The Yum repository poller is a bundled [package material](#) plugin capable of polling yum repositories for rpm packages. Go server interacts with this plugin via package material plugin interfaces. The plugin makes use of a command similar to the following to poll the server. So it does not depend on the files that yum depends on e.g. files under /etc/yum.repos.d

A given instance of polling is considered successful if repoquery returns a single package as output.

# Turn your Maven Nexus Repo into a Yum repo

Using the [Yum plugin for Nexus](#), it is possible to automatically create and publish Java artifacts as rpms using the [rpm-maven-plugin](#) and consume them on the deployment side using rpm or yum. There is a [Go webinar](#) that describes this set up.

# Repository definition

Repo URL must be a valid http, https or file URL. This plugin looks for the presence of **\$REPO\_URL/repoadata/repolmd.xml** to ascertain validity. Basic authentication (user:password@domain/path) is supported for http and https repositories.

## Package definition

In case of this plugin, the package definition is completely determined by the package spec. The package spec may be in any of the following formats. Please refer the [repoquery man page](#) for details.

```
name
name.arch
name-ver
name-ver-rel
name-ver-rel.arch
name-epoch:ver-rel.arch
epoch:name-ver-rel.arch
```

[Shell glob patterns](#) may also be used. For example, say we have a component under development getting ready for release of version 1.2.0. We cut a branch for the release and bump up the version on trunk/master to 1.3.0. Thus, a package generated by trunk/master may look like mycomp-1.3.0-b72349-noarch.rpm while that generated by branch may look like mycomp-1.2.0-b72344-noarch.rpm. Now if we have a deployment pipeline that is only interested in 1.2 series packages, the package spec needs to be mycomp-1.2.\* rather than just mycomp

## Package Metadata

The following [rpm metadata](#) is accessed by the plugin

1. BuildTime (required, automatically set by rpmbuild) - Used by the plugin to validate if the package is newer than what was last seen by Go. Go displays this field as *Modified On*.
2. Packager - Go displays this field as *Modified By*. If not provided, it is shown as anonymous
3. URL - Displayed as a *Trackback URL* by Go. **Use this as a means to trace back to the job that published the package** (within Go or outside) to the yum repository.
4. BuildHost - Displayed by Go as *Comment: Built on \$BUILDHOST*

## Published Environment Variables

The following information is made available as environment variables for tasks:

1. GO\_PACKAGE\_<REPO-NAME>\_<PACKAGE-NAME>\_LABEL
2. GO\_REPO\_<REPO-NAME>\_<PACKAGE-NAME>\_REPO\_URL
3. GO\_PACKAGE\_<REPO-NAME>\_<PACKAGE-NAME>\_PACKAGE\_SPEC
4. GO\_PACKAGE\_<REPO-NAME>\_<PACKAGE-NAME>\_LOCATION

Individual plugins may provide additional info via additional environment variables.

## Downloading RPMs

Let's say we set up a repository named ORA pointing to [http://public-yum.oracle.com/repo/OracleLinux/OL6/latest/x86\\_64](http://public-yum.oracle.com/repo/OracleLinux/OL6/latest/x86_64) and define a package gcc with a spec of gcc-4.\* and set it up as material for a pipeline. To download the package locally on the agent, we could write a task like this:

```
[go] Start to execute task: <exec command="/bin/bash" >
<arg>-c</arg>
<arg>curl -o /tmp/gcc.rpm $GO_PACKAGE_ORA_GCC_LOCATION</arg>
</exec>
```

When the task executes on the agent, the environment variables get substituted as below:

```
[go] Start to execute task: <exec command="/bin/bash" >
<arg>-c</arg>
<arg>curl -o /tmp/$GO_PACKAGE_ORA_GCC_LABEL.rpm $GO_PACKAGE_ORA_GCC_LOCATION</arg>
</exec>.
...
[go] setting environment variable 'GO_PACKAGE_ORA_GCC_LABEL' to value 'gcc-4.4.6-1'
[go] setting environment variable 'GO_REPO_ORA_GCC_REPO_URL' to value 'http://public-yum.oracle.com/repo/OracleLinux/OL6/latest/x86_64'
[go] setting environment variable 'GO_PACKAGE_ORA_GCC_PACKAGE_SPEC' to value 'gcc-4.*'
[go] setting environment variable 'GO_PACKAGE_ORA_GCC_LOCATION' to value '/tmp'
...
```

Or, to simply pass it as an argument to a deploy script on a remote server

```
<exec command="/bin/bash">
<arg>-c</arg>
<arg>ssh server "cd /to/dest/dir;deploy.sh $GO_PACKAGE_ORA_GCC_LOCATION"</arg>
</exec>
```

## Installing RPMs

For self contained packages (no external dependencies other than what is already installed on the target node), it is just enough to do:

```
rpm -U /path/to/downloaded/pkg.rpm
```

On the other hand, if the package isn't self-contained, we'd run:

```
yum install $GO_PACKAGE_ORA_GCC_LABEL
```

This would require that /etc/yum.repos.d contain the repository definitions.

## Creating and Publishing RPMs

Although the support for package as material in Go isn't concerned with how the packages are created and published, here is a short set of pointers to information on the web.

- [Building an RPM using rpmbuild and SPEC file](#)
- [Building using fpm](#)
- [Tutorial to set up a local yum repository using createrepo](#). Publishing to a yum repo simply involves uploading/copying over a new package revision at the correct location and running createrepo --update

## Notes

1. This plugin will detect at max one package revision per minute (the default interval at which Go materials poll). If multiple versions of a package get published to a repo in the time interval between two polls, Go will only register the latest version in that interval.
2. This plugin makes use of buildtime in rpm metadata to determine if a poll has returned a new result. If for some reason (e.g. timezone misconfiguration), the buildtime of pkg-1.1 is less than that of pkg-1.0, then the plugin will not register pkg-1.1 as a newer package.
3. The only way to update an rpm is to change the version or release. [Republishing](#) a different file with the same name and different buildtime won't do.
4. Package groups are not supported.
5. The [Go command repository](#) has a bunch of commands related to rpm packages.

# Task Extension

## Overview

Go supports configuring a few kinds of tasks (Nant, Ant and Rake), directly, from the configuration UI, without specifying them as a custom command. For instance, if you go to the configuration UI for a job, you'll see something like this:

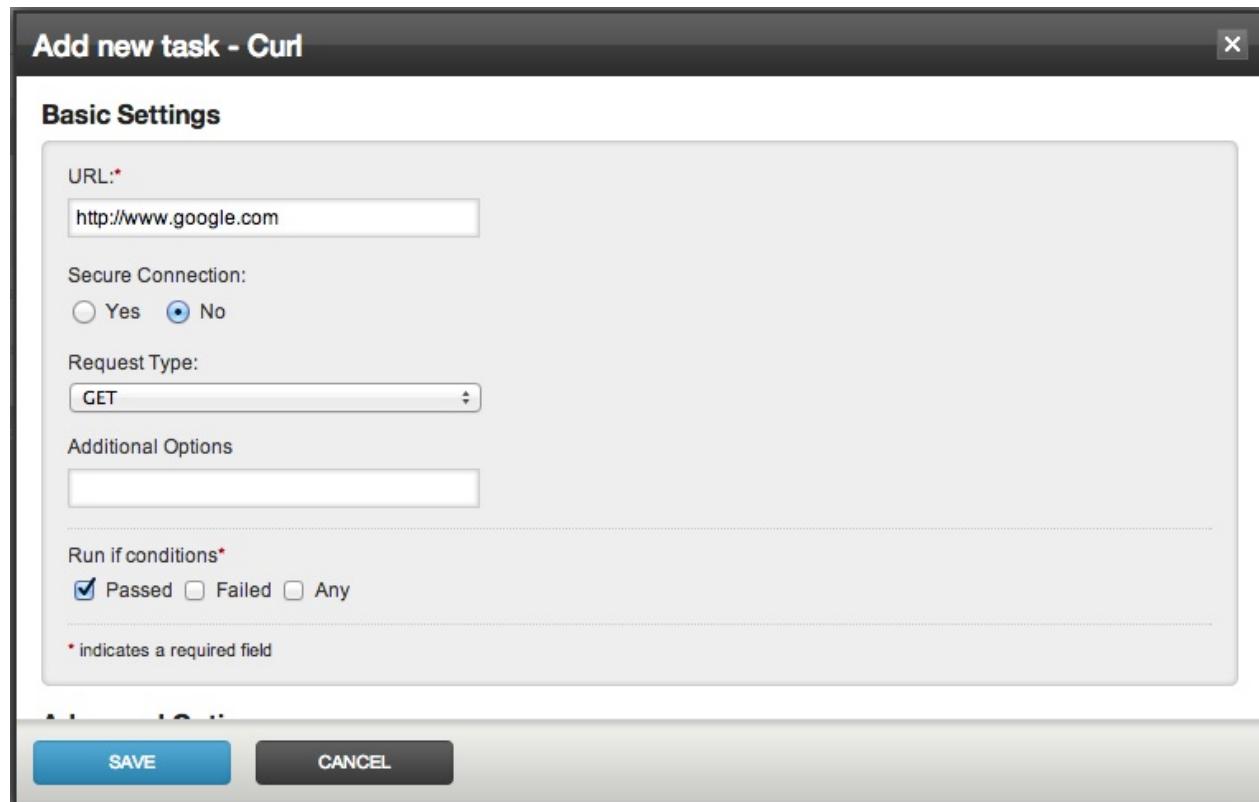
The screenshot shows the 'Tasks' tab in the Go configuration UI. At the top, there are tabs for 'Job Settings', 'Tasks', 'Artifacts', 'Environment Variables', and 'Custom Tabs'. The 'Tasks' tab is active. Below the tabs is a table with columns: Order, Task Type, Run If Conditions, Properties, On Cancel, and Remove. There is one row in the table: 'Custom Command' (Task Type), 'Passed' (Run If Conditions), 'Command: ls' (Properties), 'No' (On Cancel), and a delete icon (Remove). Below the table is a button labeled '+ Add new task ▾'. A dropdown menu is open, listing 'Ant', 'NAnt', 'Rake', 'Fetch Artifact', and 'More...'. A magnifying glass icon is at the bottom right of the dropdown.

A task plugin allows you to extend this so that you can have other tasks available here. The plugin also allows you to control the UI, as well as the data stored for this task.

For instance, you can find the source of a sample Curl plugin, [at this location](#). Assuming you have the plugin installed, you'll see that the dropdown in the job configuration UI has changed to look like this:

The screenshot shows the 'Tasks' tab in the Go configuration UI, similar to the previous one but with a plugin installed. The 'Add new task' dropdown now includes 'Curl' along with 'Ant', 'NAnt', 'Rake', and 'Fetch Artifact'. The rest of the interface is identical to the first screenshot.

When selected, the dialog box which allows you to configure details about the task looks like this:



In the configuration XML, the information entered for this task looks like this:

```
<task>
  <pluginConfiguration id="curl.task.plugin" version="1" />
  <configuration>
    <property>
      <key>Url</key>
      <value>http://www.google.com</value>
    </property>
    <property>
      <key>SecureConnection</key>
      <value>no</value>
    </property>
    <property>
      <key>RequestType</key>
      <value>-G</value>
    </property>
    <property>
      <key>AdditionalOptions</key>
      <value />
    </property>
  </configuration>
  <runif status="passed" />
</task>
```

When a build which uses the plugin runs, the output of the build looks something like

this:

```
[go] Start to execute task: Plugin with ID: curl.task.plugin.
Launching command: [curl, -G, --insecure, -o, pipelines/up42/index.txt, http://www.google.com]
Environment variables:
Name= MAVEN_OPTS Value= -Xms256m -Xmx512m
Name= GO_STAGE_COUNTER Value= 1
Name= GO_REVISION_BLAH Value= cde1e03a05170b991a92a136278c3464e4f35fe7
Name= GO_JOB_NAME Value= up42_job
Name= EDITOR Value= vim
Name= SECURITYSESSIONID Value= 186a4
... lots more environment variables ...
% Total    % Received % Xferd  Average Speed   Time     Time      Time  Current
Dload  Upload Total Spent   Left Speed
0      0      0      0      0      0      0      0 ---:---:--- ---:---:--- ---:---:--- 0
0      0      0      0      0      0      0      0 ---:---:--- 0:00:02 ---:---:--- 0
100  259  100  259      0      0    122      0  0:00:02  0:00:02 ---:---:--- 122
```

# **FREQUENTLY ASKED QUESTIONS**

---

# **OR TROUBLESHOOTING GO**

---

# Ordering of pipelines

In Go, we use two distinct types of ordering of pipelines:

- Schedule order: Chronological order in which pipelines are scheduled.
- Natural order: Chronological order of pipelines based on material modifications

In most cases the schedule order and natural order match. The user checks in and builds incrementally so the order in which builds are scheduled is the same as the relative order in which changes are checked in. Now with the "Trigger with options" functionality, it is possible to trigger a pipeline with older materials. In this case, the changes to the material as reported by the repository and the order of pipelines containing these changes are not the same

## Example

```
Order of check-ins retrieved from Hg log:
```

```
changeset: 10689:6dbb27e86dc9
branch: trunk
tag: tip
user: ShilpaG
date: Tue Apr 27 09:52:15 2010 +0530
summary: fixing twist test EnvironmentScreenPermissions

changeset: 10688:2b5b25a68117
branch: trunk
user: JJ
date: Tue Apr 27 08:45:29 2010 +0530
summary: fixing broken twist test

changeset: 10687:6f91bbb648fa
branch: trunk
user: PS
date: Mon Apr 26 15:28:16 2010 +0530
summary: #3889 - Added the twist test for the stage details actions.
```

```
Pipeline instance 1 has revision: 10687:6f91bbb648fa
Pipeline instance 2 has revision: 10689:6f91bbb648fa
Pipeline instance 3 has revision: 10688:2b5b25a68117
```

The pipeline order based on scheduling is: 1, 2, 3. This is the order in which they were triggered.

The pipeline order based on natural order: 1, 3, 2. This is because, if we look at the changes in each pipeline instance, 1 has the earliest set of revisions, 3 has the next set of revisions and 2 has the latest revisions in that particular repository (material).

The above example works when there is one material. If the pipeline has multiple materials, Go examines the timestamps of all the materials to determine which is logically the earlier pipeline instance. In this case, the earlier instance is the one that has the earliest time stamp across all the materials.

Go supports natural ordering of materials when we "Trigger with Options". The user can change the revision of all materials or where one particular material(that is likely to have broken the build) is chosen and all other materials are pinned to a particular revision(last known good revision).

# Historical Configuration

## Trace a stage run to it's config

Go provides a section on the stage details page to view the Go configuration xml used when executing a particular instance of the stage. Admin users can use this view to trace a pipeline run back to its configuration. The stage history widget which can be found on the right hand side of the stage details page has markers to indicate Go configuration changes. These markers are visible to all users.

### To navigate to the historical config:

1. Click on the stage bar of the relevant stage on the pipelines dashboard.
2. Click on the tab 'Config'.
3. Choose the stage instance from the stage history widget on the right.

Note: This tab is available to admin users only.

The screenshot shows the Go Pipelines interface for a pipeline named 'pair02' with run ID 615. The top navigation bar shows 'Pipelines » pair02 » 615 | UNLOCKED'. Below the navigation bar, there is a green progress bar labeled 'build'. The main content area displays a stage named 'build' with status 'Passed'. A red arrow points to the 'Config' tab in the stage navigation bar. Another red arrow points to the 'STAGE HISTORY' panel on the right, which lists previous configurations with markers indicating changes. The configuration XML for the stage is also visible in the main content area.

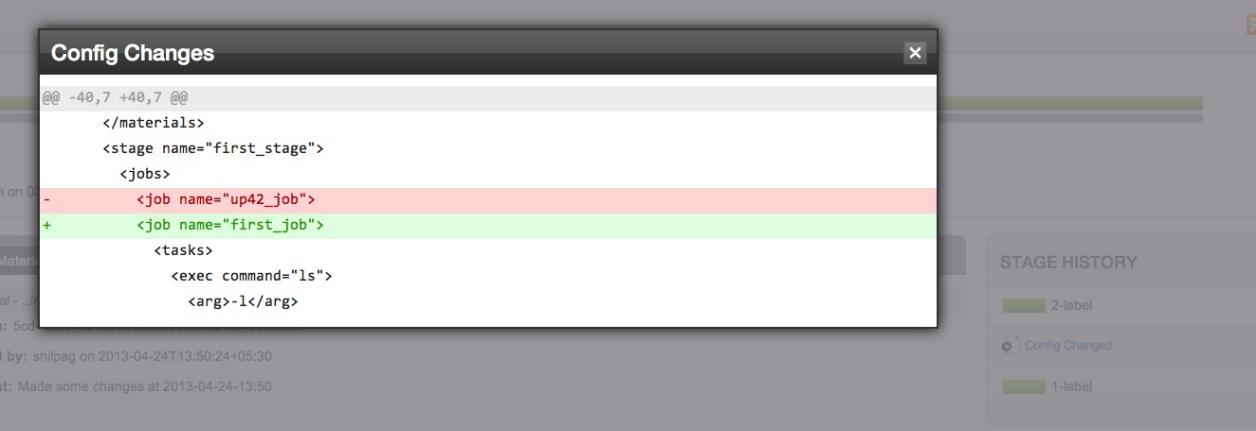
```
<arg>sss</arg>
<runif status="passed" />
</exec>
</tasks>
<resources>
<resource>DB_update</resource>
</resources>
<properties>
<property name="coverage.class" src="target/emma/coverage.xml" xpath="substring-before(coverage-class, '#')"/>
<property name="coverage.method" src="target/emma/coverage.xml" xpath="substring-before(coverage-method, '#')"/>
<property name="coverage.block" src="target/emma/coverage.xml" xpath="substring-before(coverage-block, '#')"/>
<property name="coverage.line" src="target/emma/coverage.xml" xpath="substring-before(coverage-line, '#')"/>
</properties>
</job>
</jobs>
</stage>
</pipeline>
<pipeline name="anush-2">
<params>
<param name="TAB_NAME">CustomTab</param>
<param name="TAB_PATH">cruise-output/console.log</param>
<param name="another">as</param>
<param name="a">a</param>
<param name="as">as</param>
</params>
<materials>
<svn url="https://10.4.3.20/svn/securedRepository" username="first_user" encryptedPassword="1234567890"/>
</materials>
<stage name="defaultStage">
```

Run ID	Configuration Change
848	Config Changed
847	Config Changed
846	Config Changed
845 (run 2)	Config Changed
845	Config Changed
844	Config Changed
843	Config Changed
842	Config Changed
841	Config Changed

## See what changed in the configuration

# between two stage runs

As mentioned in the previous section, the stage history widget has markers to show if configuration has changed between two stage runs. For admin users, who have the permission to view the configuration xml, the markers appear as links. Clicking on these links shows the exact difference between the configurations. The changes are shown in the same format as that of "Git Diff".



The screenshot shows a 'Config Changes' dialog box with the following XML diff output:

```
@@ -40,7 +40,7 @@
  </materials>
  <stage name="first_stage">
    <jobs>
-    <job name="up42_job">
+    <job name="first_job">
      <tasks>
        <exec command="ls">
          <arg>-l</arg>
```

Below the dialog, a 'STAGE HISTORY' sidebar is visible, showing a single entry: 'Config Changed'.

# Concurrent Modifications to Go's Configuration

---

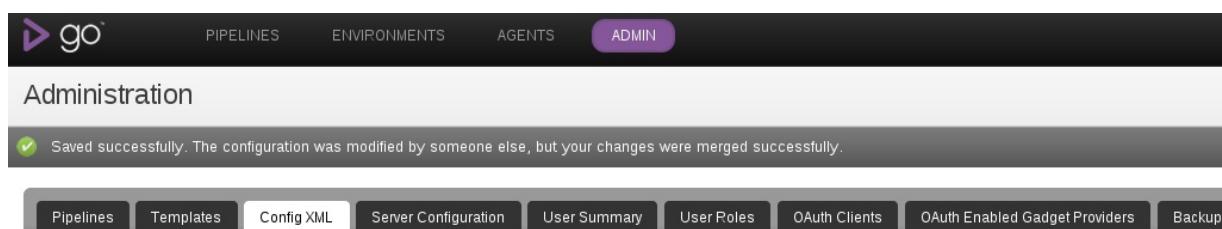
Go handles concurrent modifications to its configuration. Multiple modifications are merged and saved successfully. Modifications to the same area of configuration would result in a conflict.

Note: Configuration file is maintained in git version control system. Go leverages git's merge feature to merge changes from multiple users. As expected, concurrent changes to the same section by users would result in a conflict.

## Successful Merge

---

In case of a successful merge, user would see a success message as below:



## Merge Conflicts

---

### Handling conflict while using Config XML tab (Go Administrator)

In case of a conflict, Go provides an interface with the latest version of config along with the changes made by the user. As an example, if the same job was re-named by two users concurrently, the changes from first user would be successfully saved while the second user would see a page similar to the one displayed in the image below.

User needs to re-apply their changes displayed on the left-hand pane, to the editable version on the right and save again.

Administration

! Someone has modified the configuration and your changes are in conflict. Please review, amend and retry. [Help Topic: Configuration](#)

Pipelines Templates Config XML Server Configuration User Summary User Roles OAuth Clients OAuth Enabled Gadget Providers Backup Configuration File Path:/home/cruise/projects/cruise/server

Configuration File

The following error(s) need to be resolved in order to perform this action:

- Configuration file has been modified by someone else.

Your Changes

```
<?xml version="1.0" encoding="utf-8"?>
<cruise xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="cruise-config.xsd" schemaVersion="65">
<server artifactsdir="logs" purgeStart="4.0" purgeUpto="5.0" commandRepositoryLoca
serverId="dev-id" />
<pipelines group="PipelineGroup1">
<pipeline name="pipeline1">
<materials>
<git url="url" />
</materials>
<stage name="defaultStage">
<jobs>
<job name="test">
<tasks>
<ant target="compile test">
<runif status="passed" />
</ant>
</tasks>
</job>
</jobs>
</stage>
</pipeline>
</pipelines>
```

Last modified: less than a minute ago by anonymous Latest version of config SAVE

```
<?xml version="1.0" encoding="utf-8"?>
<cruise xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="cruise-config.xsd" schemaVersion="65">
<server artifactsdir="logs" purgeStart="4.0" purgeUpto="5.0" commandRepositoryLoca
serverId="dev-id" />
<pipelines group="PipelineGroup1">
<pipeline name="pipeline1">
<materials>
<git url="url" />
</materials>
<stage name="defaultStage">
<jobs>
<job name="run-test">
<tasks>
<ant target="compile test">
<runif status="passed" />
</ant>
</tasks>
</job>
</jobs>
</stage>
</pipeline>
</pipelines>
```

Current user changes      Saved version

## Handling conflict while using Config XML tab (Pipeline group administrator)

go PIPELINES ENVIRONMENTS AGENTS ADMIN view ▾ | Help

Administration

! Someone has modified the configuration and your changes are in conflict. Please review, amend and retry.

Pipelines Config XML

The following error(s) need to be resolved in order to perform this action:

- Configuration file has been modified by someone else.

Pipeline Groups

test\_concurrent\_edit

SAVE CANCEL

```
<pipelines group="test_concurrent_edit">
<authorization>
<view>
<user></user>
</view>
<admins>
<user></user>
</admins>
</authorization>
<pipeline name="PP">
<materials>
<git url="xcwedw" />
</materials>
```

## Handling conflict while updating configuration via other Admin tabs

If two users make similar changes to a pipeline using the ‘Edit Pipeline’ UI, the second user would see the error as displayed below.

User should backup the required changes from the page. Clicking on ‘RELOAD’ button, would discard user’s changes and reload the page with latest version of the pipeline configuration. User should re-apply his/her changes from backup and save again.

[PIPELINES](#)[ENVIRONMENTS](#)[AGENTS](#)[ADMIN](#)[Pipelines](#) » pipeline1

! Save failed. Configuration file has been modified by someone else.

[RELOAD](#)

This will refresh the page and you will lose your changes on this page.

# Why is the build broken?

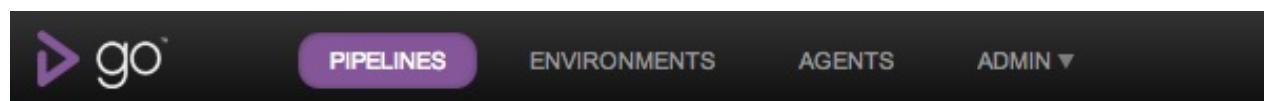
Knowing the build is broken is only the first step. Now we need to understand what caused it to break.

## Example usage

Usage: As a developer, I want to understand why the build is broken.

Let's assume that in this case, we are on a java project using JUnit as our testing library.

- If we're not already, we need to configure Go to [upload JUnit xml reports](#)
- Click on the **Pipelines** tab



- Click on the failed stage you want to investigate

A screenshot of a pipeline stage named 'acceptance'. The stage has a blue header with three arrows pointing right and the word 'acceptance'. Below the header, the text 'Label: 12.1' is shown. Underneath that, there are two buttons: 'Compare' and 'Changes'. To the right of these buttons is the text '(Triggered by tom 2 days ago) Failed: twist'. A large red arrow points from the text 'Failed: twist' towards a horizontal progress bar. The progress bar is mostly red, indicating failure. At the bottom of the stage, there are three control buttons: a play button, a plus button, and a pause button.

- Click on the failed job

Overview Pipeline Dependencies

## JOBS

▼ Failed: 6 

- firefox-1
- firefox-2
- firefox-3
- firefox-4
- firefox-5
- firefox-7

- The "Failures" sub-tab should help you diagnose what is wrong with your build

narayan-firefox/13/twist/1/firefox-1 job | failed

SCHEDULED ON:	2010-05-31T18:26:04+05:30	COMPLETED ON:	2010-05-31T19:03:36+05:30 <a href="#">more...</a>
DURATION:	00:36:32	AGENT:	birstdcrsuat02.thoughtworks.com (ip:10.4.8.2)
BUILD CAUSE:	modified by narayan		

Console Tests Failures Artifacts Materials Properties Twist [Link to this tab](#)

### ▼ Failed tests

Tests run: 10 , Failures: 2 , Not run: 0 , Time: 1545.121 seconds.

Failure ArtifactUploadFetch.scn  
Failure AgentsUIScreen.scn

Unit Test Failure and Error Details (2)

Test: ArtifactUploadFetch.scn  
Type: Failure

Message: wait timed out after THREE\_MINUTES for: Wait for pipeline: [pipeline-artifa

JOB HISTORY

- narayan-firefox/13/twist/1/firefox-1 29 days ago
- narayan-firefox/12/twist/1/firefox-1 about 1 month ago
- narayan-firefox/11/twist/1/firefox-1 about 1 month ago
- narayan-firefox/10/twist/1/firefox-1 about 1 month ago
- ✖ narayan-firefox/9/twist/3/firefox-1 about 1 month ago

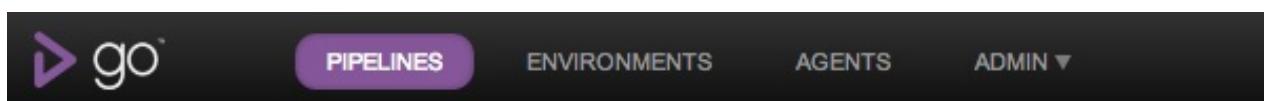
- If you need more information, the "Console" sub-tab contains everything that was written out to the console (including extra information from Go)

# See artifacts as sub-tabs

After [uploading html reports](#), it is often useful to be able to easily view this information when trying to [understand why the build is broken](#).

## Example usage

- Suppose we have configured Go to [upload a flash video and html file and display it as a tab](#)
- Click on the [Pipelines](#) tab



- Click on the stage you want to investigate

A screenshot of a Go pipeline stage named "acceptance". The stage is currently failing, as indicated by the red progress bar. Above the bar, the text "Failed: twist" is displayed. To the left of the stage name, there is a blue link labeled "Compare | Changes ▾". Below the stage name, the text "(Triggered by tom 2 days ago)" is visible. At the bottom of the stage card, there are three control buttons: a right-pointing arrow, a plus sign inside a triangle, and a double vertical bar symbol.

- Click on the job you want to investigate

The screenshot shows a top navigation bar with two tabs: "Overview" and "Pipeline Dependencies". The "Pipeline Dependencies" tab is highlighted with a dark grey background and white text.

## J OBS

- ▶ Failed: 0
- ▶ In Progress: 0
- ▼ Passed: 2
  - build-debug
  - build-release

- Click on the tab you created

This screenshot shows the details for a successful build job named "acceptance/2.0.0.2.0.0.5068/build-debug". The job status is "passed". Below the title, there are several build statistics:

SCHEDULED ON:	2010-06-29T16:28:57+05:30	COMPLETED ON:	2010-06-29T16:57:41+05:30 <a href="#">more...</a>
DURATION:	00:28:25	AGENT:	b1rstdcrsuat03.thoughtworks.com (ip:10.4.8.3)
BUILD CAUSE:	triggered by cruise/5068/dist/1		

Below the statistics is a navigation bar with several tabs: "Console", "Tests", "Failures", "Artifacts", "Materials", "Properties", "Recording", and "JOB HISTORY". The "Recording" tab is circled in red.

- Clicking on the tab will load the page, which will start the video!
- To view the content in a new window, just click "Download Recording"

# Saving properties about a build

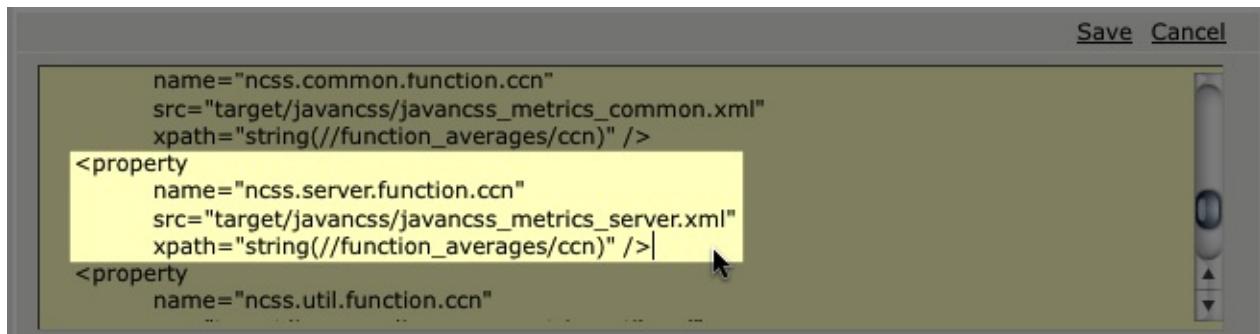
When building your code, there can be quite a bit of extra information that you are interested in. For example, you might run [EMMA](#) on your code in order to log code coverage. With properties, you can save this information, and even look at the history of a property (by way of an exported spreadsheet).

## Example usage

Usage: As a developer, I want to save the average [cyclomatic complexity](#) of a function (pulled from [JavaNCSS](#)).

For this example, we're going to take the information out of "target/javancss/javancss\_metrics\_util.xml"

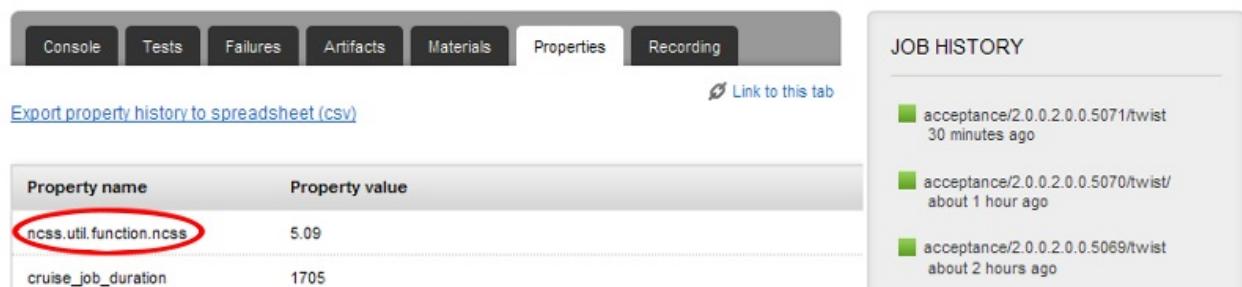
- On the [Administration Tab](#), edit the job that should generate the properties
- Ensure the following "properties" block is in the job configuration



The screenshot shows a configuration interface for a Jenkins job. At the top right are 'Save' and 'Cancel' buttons. Below them is a code editor containing XML configuration. A yellow highlight box surrounds the following code block:

```
name="ncss.common.function.ccn"
src="target/javancss/javancss_metrics_common.xml"
xpath="string(//function_averages/ccn)" />
<property
    name="ncss.server.function.ccn"
    src="target/javancss/javancss_metrics_server.xml"
    xpath="string(//function_averages/ccn)" />
<property
    name="ncss.util.function.ccn"
```

- Now, after that job has run, you should have extra properties information on the [Job Details](#) page



The screenshot shows the 'Properties' tab of a Jenkins job's details page. At the top, there are tabs for Console, Tests, Failures, Artifacts, Materials, Properties (which is selected), and Recording. Below the tabs is a link to 'Export property history to spreadsheet (csv)'. To the right is a 'JOB HISTORY' section with three entries:

Property name	Property value
ncss.util.function.ncss	5.09
cruise_job_duration	1705

The 'ncss.util.function.ncss' entry is circled in red.

JOB HISTORY

- acceptance/2.0.0.2.0.0.5071/twist 30 minutes ago
- acceptance/2.0.0.2.0.0.5070/twist/ about 1 hour ago
- acceptance/2.0.0.2.0.0.5069/twist/ about 2 hours ago

- You can export the property history as a CSV file

# Using Environment Variables in Go

## Standard Go environment variables

Environment Variable	Description	Example value
GO_SERVER_URL	Base URL for the Go server (including the context root)	<a href="https://127.0.0.1:8080">https://127.0.0.1:8080</a>
GO_ENVIRONMENT_NAME	The name of the current environment. This is only set if the environment is specified. Otherwise the variable is not set.	Development
GO_PIPELINE_NAME	Name of the current pipeline being run	main
GO_PIPELINE_COUNTER	How many times the current pipeline has been run.	2345
GO_PIPELINE_LABEL	Label for the current pipeline. By default, this is set to the pipeline count (this can be set to a <a href="#">custom pipeline label</a> )	1.1.2345
GO_STAGE_NAME	Name of the current stage being run	dev
GO_STAGE_COUNTER	How many times the current stage has been run	1
GO_JOB_NAME	Name of the current job being run	linux-firefox
	Username of the user that	

GO_TRIGGER_USER	<p>triggered the build. This will have one of three possible values</p> <ul style="list-style-type: none"> <li>• anonymous           <ul style="list-style-type: none"> <li>- if there is no security</li> </ul> </li> <li>• username of the user, who triggered the build</li> <li>• changes, if SCM changes auto-triggered the build</li> <li>• timer, if the pipeline is triggered at a scheduled time</li> </ul>	<span>changes</span>
GO_DEPENDENCY_LABEL_\${pipeline name}	<p>The label of the upstream pipeline (when using <a href="#">dependent pipelines</a>)</p>	1.0.3456
GO_DEPENDENCY_LOCATOR_\${pipeline name}	<p>The locator of the upstream pipeline (when using <a href="#">dependent pipelines</a>), which can be used to create the URL for RESTful API calls</p>	upstream/1.0.3456
GO_REVISION	<p>The current source control revision being run (when using only one material)</p>	123
	<p>If you are using more than one material in your pipeline, the revision for each material is available. The environment variable is named with the</p>	

GO_REVISION_{material name or dest}	material's "materialName" attribute. If "materialName" is not defined, then "dest" directory is used. Non alphanumeric characters are replaced with underscores ("_").	123
GO_TO_REVISION	If the pipeline was triggered with a series of source control revisions(say 121 to 123), then this environment variable has the value of the latest revision (when using only one material). This is always same as GO_REVISION.	123
GO_TO_REVISION_{material name or dest}	If you are using more than one material in your pipeline, the 'to' revision for each material is available. The environment variable is named with the material's "materialName" attribute. If "materialName" is not defined, then "dest" directory is used. Non alphanumeric characters are replaced with underscores ("_").	123
GO_FROM_REVISION	If the pipeline was triggered with a series of source control revisions(say 121 to 123), then this	121

	environment variable has the value of the oldest revision (when using only one material)	
GO_FROM_REVISION_\${material name or dest}	If you are using more than one material in your pipeline, the 'from' revision for each material is available. The environment variable is named with the material's "materialName" attribute. If "materialName" is not defined, then "dest" directory is used. Non alphanumeric characters are replaced with underscores ("_").	121

## Use current revision in a build

---

It is often useful to use the current version control revision number in your build. For example, you might want to use the svn version number in the name of your binary for tracing purposes. Go makes much of this information available to your build scripts as environment variables.

### Example usages

#### One material

For this example, we are going to assume we are using a single [Subversion](#) repository for our source control system and we have a job set up to call the ant target "dist".

- Add the following target to your ant build.xml
- Now, when Go runs the 'my-app' pipeline on revision 123, the file deploy-123.txt will be created, with the following content:

## Multiple materials

For this example we are going to assume we are using a [Subversion](#) repository containing the code and a [Mercurial](#) repository containing configuration scripts.

- Ensure the pipeline materials look like this
- Add the following target to your ant build.xml
- Now, when Go runs the 'my-app' pipeline with the code at revision '123' and the configuration at revision '59cab75ccf231b9e338c96cff0f4adad5cb7d335', the file deploy-123.txt will be created with the following content:

## Pass environment variables to a job

You can specify variables for Environments, Pipelines, Stages and Jobs. If a variable is specified more than once, the most specific scope is used. For example if you specify variable FOO='foo' for an environment, and FOO='bar' for a Job, then the variable will have the value 'bar' when the job runs.

## Setting variables on an environment

You can add variables to an environment by editing the configuration of the environment. Click on the name of the environment to edit configuration.

The screenshot shows a dialog box titled "Environment Variables". It contains two input fields: "version" set to "12.3" and "environment" set to "UAT". There is a "Save" button at the bottom left and a "Cancel" button at the bottom right.

version	=	12.3	X
environment	=	UAT	X

+ Add

SAVE CANCEL

You specify variables on an environment in the Config XML by adding an <[environmentvariables](#)> section to the environment definition.

```
<environment name="UAT">
  <environmentvariables>
```

```
<variable name="FOO">
    <value>bar</value>
</variable>
<variable name="MULTIPLE_LINES">
    <value>Variable values can have
multiple lines (assuming that your operating system supports this correctly)
</value>
</variable>
<variable name="COMPLEX">
    <value><! [CDATA[<complex
values>]]>
    </value>
</variable>
</environmentvariables>
<agents />
<pipelines />
</environment>
```

You can add variables for a job by editing the job configuration.

The screenshot shows the Jenkins job configuration interface. At the top, there are tabs for Job Settings, Tasks, Artifacts, Environment Variables (which is currently selected), and Custom Tabs. Below the tabs, the 'Environment Variables' section is displayed. It contains two entries: 'ruby\_version' with a value of '1.8.7' and 'mysql\_version' with a value of '5.5'. Each entry has a delete button ('x') to its right. At the bottom left of the form, there is a green '+' button labeled 'Add'.

Name	Value
ruby_version	= 1.8.7
mysql_version	= 5.5

You specify variables on an job in the Config XML by adding an [<environmentvariables>](#) section to the job definition.

```
<job name="my-job">
    <environmentvariables>
        <variable name="FOO">
            <value>bar</value>
        </variable>
        <variable name="MULTIPLE_LINES">
            <value>Variable values can have
multiple lines (assuming that your operating system supports this correctly)
        </value>
        </variable>
        <variable name="COMPLEX">
            <value><! [CDATA[<complex
values>]]>
            </value>
        </variable>
    </environmentvariables>
```

```
</environmentvariables>  
...  
</job>
```

## Using environment variables in task

---

You can access these environment variables to construct versioned artifacts or to store properties on the current build. For example the following snippet of an ant file shows how to access Go variables:

```
<property environment="go" />  
<target name="all">  
    <echo message="Building all!" />  
    <echo message="GO_SERVER_URL: ${go.GO_SERVER_URL}" />  
    <echo message="GO_PIPELINE_NAME: ${go.GO_PIPELINE_NAME}" />  
    <echo message="GO_PIPELINE_COUNTER: ${go.GO_PIPELINE_COUNTER}" />  
    <echo message="GO_PIPELINE_LABEL: ${go.GO_PIPELINE_LABEL}" />  
    <echo message="GO_STAGE_NAME: ${go.GO_STAGE_NAME}" />  
    <echo message="GO_STAGE_COUNTER: ${go.GO_STAGE_COUNTER}" />  
    <echo message="GO_JOB_NAME: ${go.GO_JOB_NAME}" />  
    <echo message="GO_REVISION: ${go.GO_REVISION}" />  
</target>
```

CRUISE\_XXX variables are deprecated since Go 2.0. Please use GO\_XXX instead of CRUISE\_XXX (For example: GO\_SERVER\_URL instead of CRUISE\_SERVER\_URL).

# Releasing into an environment

---

One of the most useful aspects of having your build mapped as a pipeline, is being able to know exactly what is in a particular environment. For example, you might have a User Acceptance Testing environment into which you want Go to automatically deploy your binary. Due to process restriction within your company, you might want to manually install a binary yourself, but have Go still retain the information of what is currently released.

## Example usages

---

### Automatically deploy to UAT

For this example, we'll assume that there is already an ant task defined in your build that will take a binary and deploy it to your environment. A separate task will verify the install was successful. If it is not, a task will run to rollback the deployment. We will also assume that earlier in the pipeline there is a **dist** stage with a **create-installers** job that will have already created the binary for us to use.

- [Add a new stage](#) named **UAT** with a job named **deploy**
- [Ensure that the \*\*UAT\*\* stage is manual](#)
- Ensure the following task block is in the **deploy** job configuration

When you are ready to deploy something into the UAT environment...

- Navigate to the [pipeline activity](#) page
- Find the check-in you want to deploy
- Click on the manual transition into the **UAT** stage



- When the deploy is successful, the stage will be green and the UAT environment will contain the selected check-in

dist

uat



- When the deploy fails for some reason, the stage will be red and the UAT environment will contain the original check-in

## Manually deploy to production

For this example, we'll assume that there is a known way to rollback to a previous installation. We will also assume that earlier in the pipeline there is a **dist** stage with a **create-installers** job that will have already created the binary for us to use.

- Add a new stage named **production** with a job named **deploy**
- Ensure that the **production** stage is manual
- Ensure there is no task block in the **deploy** job configuration

When you are ready to deploy something into the production environment...

- Navigate to the [pipeline activity](#) page
- Find the check-in you want to deploy
- Click on the details link of the **dist** stage

dist



- Download the installer binary in the artifacts tab

## cruise/2.0.0.5066-4584803c431f47dd8f1

SCHEDULED ON: 2010-06-29T11:23:53+05:30  
DURATION: 00:10:45  
BUILD CAUSE: modified by jmm & jb

Console

Tests

Failures

Artifacts

Materials

(+) Expand All (-) Collapse All

cruise-output

pkg

- [cruise-agent-2.0.0-11331-osx.zip](#)
- [cruise-agent-2.0.0-11331-setup.exe](#)
- [cruise-agent-2.0.0-11331.deb](#)
- [cruise-agent-2.0.0-11331.noarch.rpm](#)
- [cruise-agent-2.0.0-11331.zip](#)
- [cruise-agent-2.0.0.zip](#)
- [cruise-server-2.0.0-11331-osx.zip](#)
- [cruise-server-2.0.0-11331-setup.exe](#)
- [cruise-server-2.0.0-11331.deb](#)

- Manually install the binary into production
- If there are issues, manually rollback to the last known good installation
- If everything seems to be working correctly, click on the manual transition into the **production** stage

# What has changed in the current version?

---

When updating your testing environments to a new version, it is useful to know what changes have been made since it was last updated. Since there is currently no way to get this information in Go automatically, there are some extra steps we must take.

## Example usage

---

For this example, we'll assume that there is a manual "UAT" stage will automatically deploy and install an executable on your user acceptance testing machine.

- On the [Pipelines](#) page, click on the name of your pipeline



- Now that you're on the [pipeline Activity](#) page, you can see exactly how far each check-in has gotten in your pipeline
- Find the check-in that's currently in UAT. In this example, it has the pipeline label of **2.0.0.5077**



- For every check-in earlier than the one in UAT, click to see the comments

# Dependency Management

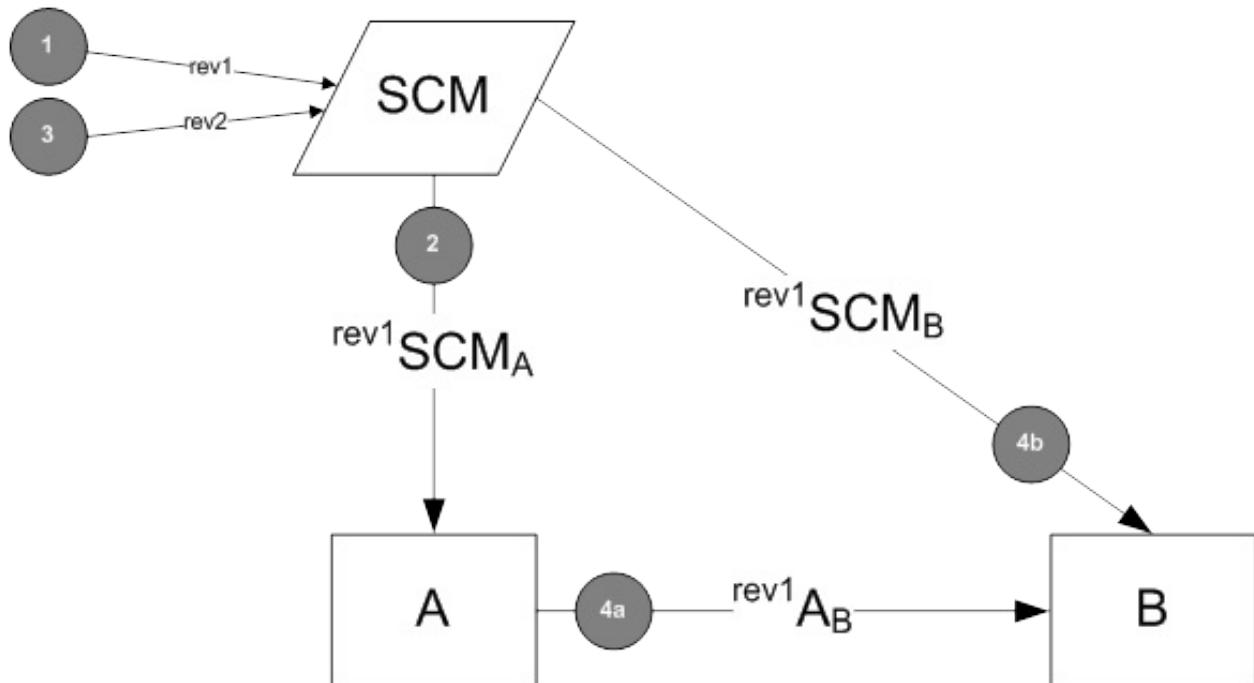
When you have non-trivial dependency pipeline chains, you may have concerns about how dependent pipelines and materials interact. For example, code and tests are checked in as part of the same commit. But code is built and tested in sequence, so the same material version has to be used for pipelines that build and test your code. This section covers some Dependency Management concepts and how Go handles certain complex scenarios.

## Propagate material revision throughout the dependency chain

If you have frequent material updates coupled with long running dependent pipelines sharing that same material, you may encounter situations where the revision that triggered the dependency chain is no longer the latest revision. Go keeps track of what revision triggered a dependency chain and ensures that the same version propagates throughout all members of that chain. This helps ensure that all artifacts generated as part of that build share a common revision.

### Example

Consider the following dependency chain:



## Legend

- **SCM** : Repository
- **rev1, rev2** : Check-ins to the repository
- **A** : Pipeline for development build
- dependent **B** : Pipeline for acceptance tests

## How it works

- Code is checked in to SCM (rev1)
- Development build (A) is triggered by the check-in
- There is another check-in to SCM (rev2)
- Development build completes (with rev1) and triggers acceptance tests (B)
- Here's where dependency management comes in. Go is smart enough detect that rev1 originally triggered the build and ensures that the acceptance tests check-out that revision (rev1) and not the latest revision (rev2). In this situation, Go ensures that the appropriate version of acceptance tests are run against the appropriate version of the development build.

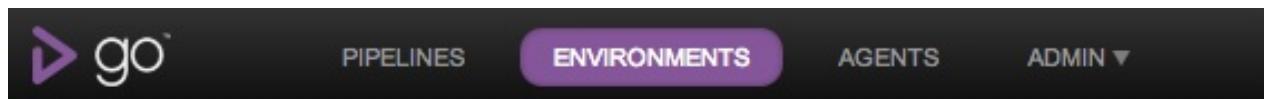
# Discover what's in an environment

Before [deploying something into production](#), it is often useful to know what is currently there.

## Example usage

For this example, we will assume we have a stage name "production" that will automatically deploy a binary onto a production server

- Start at the [Environments](#) page



- Click on the name of your "production" stage

A screenshot of the Go application interface showing the 'Production' environment details. The card includes:

- A green progress bar with a red oval highlighting its middle section.
- A button labeled '▶ +'

- The [Stage Details](#) page will show every time Go has deployed your application

# Deploy specific revisions of the materials to an environment

---

Go allows you to hand pick which revision of your materials you would like to deploy to your environment. This is a very common requirement on larger projects which have multiple materials in their deployment pipeline. Sometimes you may wish to have control over which revision of the application is deployed to a particular environment (say UAT).

## Select specific revisions of materials to deploy

Consider the case where a deployment pipeline 'deploy\_bookstore' has 2 materials - Material 'svn' and upstream pipeline 'bookstore'. It is very common to know that label, say "3.4-RELEASE" of the dependent pipeline 'bookstore' is stable. All the changes that you want right now in your UAT environment are made to material 'svn'. In such a scenario when you deploy "deploy\_bookstore" to UAT, you might always want to select label "3.4-RELEASE" of pipeline 'bookstore' and the latest (or a known revision specified by your developer) of material 'svn'.

Once there are any new changes to any of the materials, Go will indicate to the user that newer revisions are available to deploy. You could use this information and deploy a custom build with hand picked revision or deploy the latest available revision.

Steps to select the revisions of materials you want to deploy

- Navigate to the Environments page and locate the specific deployment pipeline you are interested in.
- Click on "Deploy Specific Revision".

**Production**

Label: 45  
(Triggered by smith 3 months ago)  
Passed: smoke

Material Revision Check-in/trigger

application	<b>application/45/ ft/1</b>	2 days ago
-------------	-----------------------------	------------

▶ ▶+

- This gives you the list of available revisions for each material
- Click on the "Revision to Deploy" search box. This will list latest 5 revisions/labels of your materials ordered by time of check-in (latest check-in on top)

**Production - Deploy**

Materials Environment Variables

Pipeline: application  
Dest: not-set  
Date: 4 months ago  
User: smith  
Comment: Committing deploy scripts  
Currently Deployed: application/45/ft/1  
Revision to Deploy:

application/54/ft/1	3 months ago
application/53/ft/3	3 months ago
application/50/ft/1	3 months ago
application/52/ft/2	3 months ago
application/53/ft/2	4 months ago

DEPLOY CHANGES CLOSE

- Select the revisions of all the materials that you would like to pick for deployment. You can search for the revision you want by
  - revision hash/pipeline label

- check-in comment
- user
- If you do not select a specific revision of a material, then the currently deployed revision will be retained.
- Before clicking on "Deploy Changes", check the "To Deploy" column to verify which revision will be deployed.
- Click "Deploy Changes" to start the deployment.

## Why is the 'Deploy Changes' button disabled?

There are 3 reasons this can happen

- There is a deployment in progress, so another one cannot be started
- Your deployment pipeline is operating in [locked](#) mode
- You do not have sufficient permissions to operate on that pipeline

## Deploying the latest of all materials

If you always want to have the latest of all materials deployed to your environment, then this is how you can use Go to do it.

- Click on deploy latest
- This will trigger the deployment pipeline
- This will pick up the latest available revision of your materials at the time the pipeline is scheduled

## Using passwords while deploying

- You can set secure variables in Go that gets passed along as environment variables to the executing task. You can use this feature to pass passwords to deploy scripts. For e.g., you can define a secure variable named 'DB\_DEPLOY\_PASSWORD' and the DB password as its value. This value will be encrypted by Go and passed along to the task.

## » module\_3\_production\_deploy

General Options Project Management Materials Stages Environment Variables **Parameters**

**Environment Variables** ?

Name	Value
DB_DEPLOY_HOST	= 10.20.30.40 <span style="color: red;">x</span>
	= <span style="color: red;">x</span>

+ Add

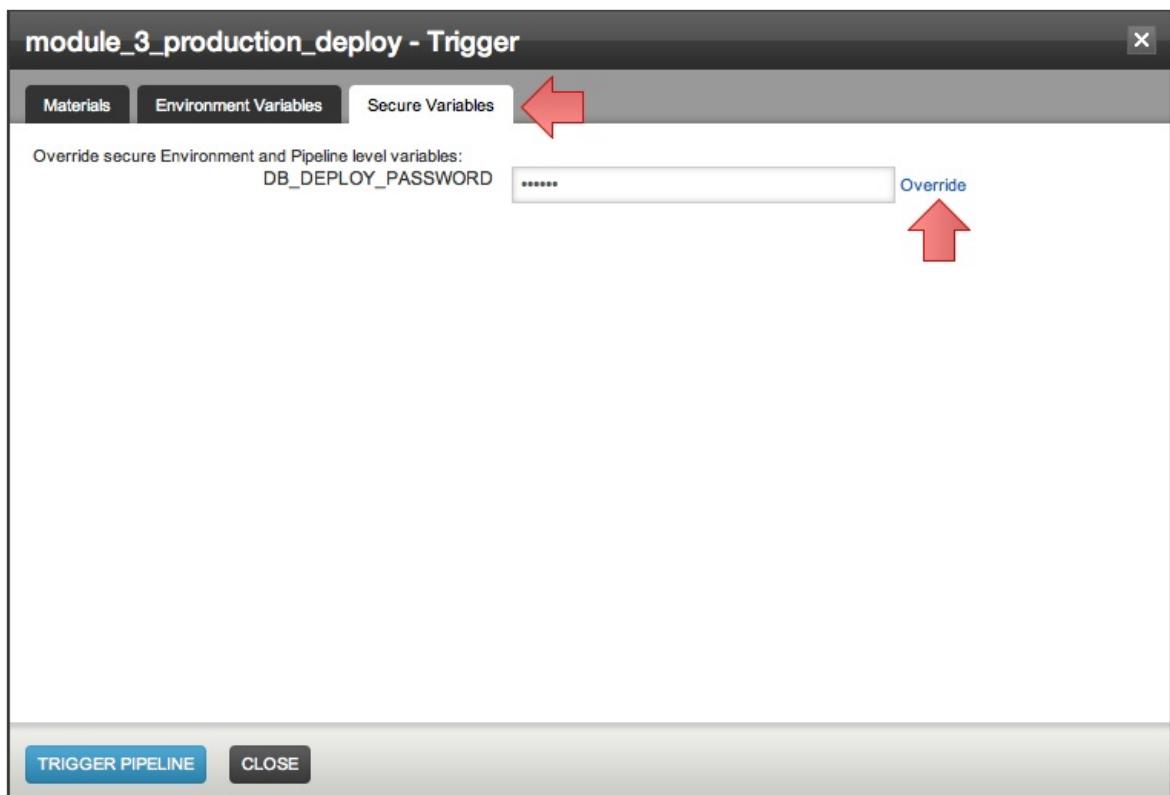
**Secure Variables** ?

Name	Value
DB_DEPLOY_PASSWORD	= ..... <span style="color: red;">x</span>

+ Add

**SAVE** **RESET**

- Also, you can override secure variables when you use the 'Trigger With Options' feature.



# Setting up a new agent by cloning an existing agent

---

## Clone the agent

---

An easy way to set up a new agent is to clone an existing one which is already set up and is known to work correctly. You can clone an agent in a couple different ways:

- Copy the entire agent directory structure to a new location (can be on the same machine or on a different one).
- If you use a virtual machine to run your agent, clone the entire VM.

## Remove duplicate GUID

---

Once the agent has been cloned, you'll need to delete the GUID file so that Go server does not confuse the new agent for the old one. The GUID file can be found at *[agent working dir]/config/guid.txt* - delete this file from the new agent.

## Approve the new agent

---

You may now start your agent and it should appear under the Agents tab on the Go server. Approve it as you would any new agent and you're ready to go.

# OAuth Overview

---

Go implements the OAuth protocol to authorize third party application's (client's) request to access data on the Go server.

## What is OAuth?

---

OAuth is an open-source specification for building a framework for allowing a third-party app (the “client”) to access protected resources from another application (the “provider,” or “resource owner”) at the request of a “user” of the client app. Oauth allows the user to enter his user credentials (ex. username and password) only to the provider app, which then grants the client app permission to view the protected resources on behalf of the user.

### Common terms:

- **Provider/Resource Owner** – the app that hosts the protected resource. An example is Twitter which uses OAuth as the protocol for all its clients. In the context of this document, Go is the provider/resource owner.
- **Client** – the app that requests to see the resource data on behalf of the user. Any Twitter client that shows tweets is an example of this. An HTML gadget that displays the status of all Go pipelines running on a server is also an example of a client.
- **User/end user** – the entity who initiates the OAuth flow to allow the client to access protected data from the provider.
- **Client id/client secret** – Often, provider apps will maintain a list of clients that are allowed to access their data. Client apps can be identified in a number of ways, including with an id and a secret.

## OAuth Authorization Workflow

---

An overview of the basic OAuth workflow can be found at [Beginner's guide to OAuth](#).

## Manage OAuth Clients

---

### Create a new OAuth client

Before any third-party application can use Go gadgets, it needs to be registered in Go as an OAuth client.

- Login as an administrator to Go.
- Navigate to the **Admin page**, then the **OAuth Clients** tab.
- Click the **New OAuth Client** button.
- Fill in the **Name** and **Redirect URL** for the third-party application. The redirect URL is where Go will send the end-user to once the authorization process is complete.
- You'll be presented with a summary of the newly registered application. Use the provided Client ID and Secret in the third-party application to enable OAuth communications with Go.

## Edit an existing OAuth client

If you've already registered an OAuth Client, but want to change its name or redirect URL, here's how:

- Login as an administrator to Go.
- Navigate to the **Admin page**, then the **OAuth Clients** tab.
- Locate the Client you want to modify and click the **Edit** link next to it.
- Edit the necessary fields and click the **Update** button to save your changes.

## Delete an existing OAuth client

If you want to un-register/delete an OAuth Client (prevent it from accessing Go via OAuth), here's how:

- Login as an administrator to Go.
- Navigate to the **Admin page**, then the **OAuth Clients** tab.
- Locate the Client you want to delete and click the **Destroy** link next to it.
- Confirm the deletion in the popup box.

## Consume Go Gadgets

---

If you are a third-part client developer and want to consume Go gadgets, the following sections will provide you an overview of what you need to do to consume gadgets using OAuth.

### Request for authorization code

Your client needs to contact Go server for an authorization code using the client Id and client secret. Go verifies that the requesting application has been registered with it.

Send a request to: <https://your-go-server.com:8154/oauth/authorize> with the following query parameters:

Query Parameter	Description
client_id	(required) The client identifier for your application.
redirect_uri	(required) URL where the user should be redirected to after access to the service is granted. This uri can also include url-encoded query parameters
response_type	(required) The type of response. Should always be 'code' in this case

### Example Request:

```
https://www.your-go-server.com:8154/oauth/authorize?redirect_uri=http://www.my_redirect_uri
```

If you type the above request on a browser, you should see a form asking you to authorize the client to access the host application on your behalf. Check the check box and submit.

Your browser is redirected to your redirect URI and you should now see your authorization code as the “code” parameter. Save this code, you will need it for the next step.

### Example Response:

```
http://www.my_redirect_uri.com?code=26a7dea5e7e121be5ad5832a4a5b09d505c234c7625de3f3*
```

## Get access token

For this step you'll need to send a POST request to /oauth/token with the following key/value pairs as form data:

Form Data Parameter	Description
---------------------	-------------

code	(required) This is the authorization code that you got from the previous step.
grant_type	(required) Should be 'authorization-code' for this request.
client_id	(required) The client identifier for your application.
client_secret	(required) The client secret for your application.
redirect_uri	(required) URL where the user should be redirected to after access to the service is granted. This uri can also include url-encoded query parameters

### Example Request (in curl) :

```
curl https://www.your-go-server:8154/go/oauth/token -d "code=26a7dea5e7e121be5ad58c&grant_type=authorization_code&client_id=your-client-id&client_secret=your-client-secret&redirect_uri=https://www.your-go-server:8154/go/&access_type=offline"
```

The response to the above POST request will be a JSON containing your access token, and its expiration time in seconds

### Example Response:

```
{ :access_token => f180f7bb68d38531aac2f49e5b0cac0c5ed5ced9b72842a429e783747e819664, :token_type => "bearer", :expires_in => 3600, :scope => "" }
```

## Use the access token

Now you are ready to query data from the Go server.

### Example Request:

```
curl -H 'Authorization: Token token="f180f7bb68d38531aac2f49e5b0cac0c5ed5ced9b72842a429e783747e819664"
```

# OAuth 2.0

---

OAuth is an open-standard that enables a user to share private protected resources (e.g. photos or financial records) which she stores on one site (or application) with another one of her commonly used sites or applications without asking her to share any passwords between the two sites. OAuth is quickly becoming a widely adopted standard; for example, Yahoo, Facebook and Twitter have all adopted OAuth.

In the context of ThoughtWorks Studios applications, the application data is the private protected resource. For example, if someone had configured Go and Mingle integration, a Go user will only be allowed to see Mingle card information in Go that he would normally be allowed to see in Mingle. That is, when Go shows Mingle data in its pages, the Mingle authorization rules are not relaxed to allow all members of that Go pipeline group to automatically see the card activity for any Mingle project, rather he will only be able to see data from Mingle projects for which he has access. The same can be said when Mingle shows Go data in its pages. It should also be possible to maintain this principle when showing Go or Mingle content in a 3rd party, non-ThoughtWorks Studios context.

In order to make this work, Go and Mingle - and in the future all Studios server-based applications - use OAuth 2.0 (v9) as a means of allowing a user of one application to establish his identity in the other application, while also granting the appropriate data access for that user. Both applications are OAuth providers and both applications run a gadget rendering server capable of acting as an OAuth client. Currently, OAuth is only used for gadget-related communication, but we plan on expanding what data can be made available via OAuth in the future.

Below is a series of movies we've made that explain how OAuth works. Part 1 covers the basics and is likely enough for most users. Parts 2-4 get into the more technical details of how the protocol works.

# OpenSocial gadgets

---

ThoughtWorks Studios products use the [OpenSocial](#) gadget stack to provide UI integrations. This provides a secure, standards-based platform for displaying one application's content in another application. Part of this infrastructure is that Go is itself an OpenSocial gadget rendering server. In general, a gadget is simply a piece of content from another application, such as showing a Mingle card activity table inside of Go. If you use iGoogle and have customized your home page, you have worked with OpenSocial gadgets.

Most OpenSocial gadgets are publicly available content that do not require authentication and authorization. However, some gadgets, such as those published by the ThoughtWorks Studios products Go and Mingle, do require authorization. To accomplish this, Go's gadget rendering server supports OAuth 2.0 services.

Enabling Go for OAuth 2.0 enabled gadgets does require the Go administrator to take [extra configuration steps](#).

If you are simply looking to configure the display of Mingle card activity gadgets in Go, please skip straight to the [instructions that are specific to showing Mingle gadgets in Go](#).

A gadget rendering server with OAuth 2.0 capabilities similar to what Go provides would be capable of showing ThoughtWorks Studios gadgets. That is, if iGoogle were to start supporting OAuth 2.0 in conjunction with its gadget support, and your Go instance was on a public server, it would be possible to display ThoughtWorks Studios gadgets on your home page. This is not currently possible so look for more on this from us in the future.

# Re-running Job(s)

You may sometimes encounter situations where you want to re-run only a subset of jobs within a stage rather than the entire stage or pipeline. Examples of such scenarios include:

- Environmental problems on a particular agent caused a job to fail
- Unsuccessful build deployment to one (or more) servers within a cluster of servers

## To re-run a job

- Navigate to the **Stage Details** screen of the stage who's job you want to re-run.
- Click on the **Jobs** tab.
- Check the job(s) you want to re-run and click the **Rerun** button.

The "Rerun" option is only available for stages that have completed. You cannot re-run jobs for stages that are still running.

The screenshot shows the Stage Details interface. At the top, there are tabs for Overview, Pipeline Dependencies, Materials, Jobs (which is selected), and Tests. Below the tabs is a large red arrow pointing to the 'RERUN' button. To the right of the table is a 'STAGE HISTORY' section containing several status entries. One entry for 'fourthJob' is highlighted with a red arrow, indicating it has been re-run.

Name	Result	State	Duration	Agent
<input checked="" type="checkbox"/> fourthJob	Failed	Completed	12 seconds	blrstdcrspbs02.thoughtworks.com(10.4.7.82)
<input type="checkbox"/> firstJob	Passed	Completed	8 seconds	blrstdcrspbs01.thoughtworks.com(10.4.7.81)
<input type="checkbox"/> secondJob	Passed	Completed	10 seconds	blrstdcrspbs02.thoughtworks.com(10.4.7.82)
<input type="checkbox"/> thirdJob	Passed	Completed	15 seconds	blrstdcrspbs01.thoughtworks.com(10.4.7.81)

Stage	Run Number
fourthJob	22
firstJob	21
secondJob	20
thirdJob	19
fourthJob	18
firstJob	17
secondJob	16 (run 2)
thirdJob	16

Job re-runs are denoted by a circular arrow overlay on the stages and jobs that have been re-run:

The screenshot shows the Stage Details interface with the same layout as the previous one. A red arrow points to the 'firstJob' row in the table, which now has an 'Active' status and a circular arrow icon next to it, indicating it is currently being re-run. The 'STAGE HISTORY' section on the right shows the same list of runs, with the first entry for 'fourthJob' also having a circular arrow icon, indicating it was part of a re-run.

Name	Result	State	Duration	Agent
<input type="checkbox"/> fourthJob	Active	Scheduled		Not yet assigned
<input type="checkbox"/> firstJob	Passed	Completed	8 seconds	blrstdcrspbs01.thoughtworks.com(10.4.7.81)
<input type="checkbox"/> secondJob	Passed	Completed	10 seconds	blrstdcrspbs02.thoughtworks.com(10.4.7.82)
<input type="checkbox"/> thirdJob	Passed	Completed	15 seconds	blrstdcrspbs01.thoughtworks.com(10.4.7.81)

Stage	Run Number
fourthJob	22 (run 2)
firstJob	21
secondJob	20
thirdJob	19
fourthJob	18
firstJob	17

# Go unable to poll for changes

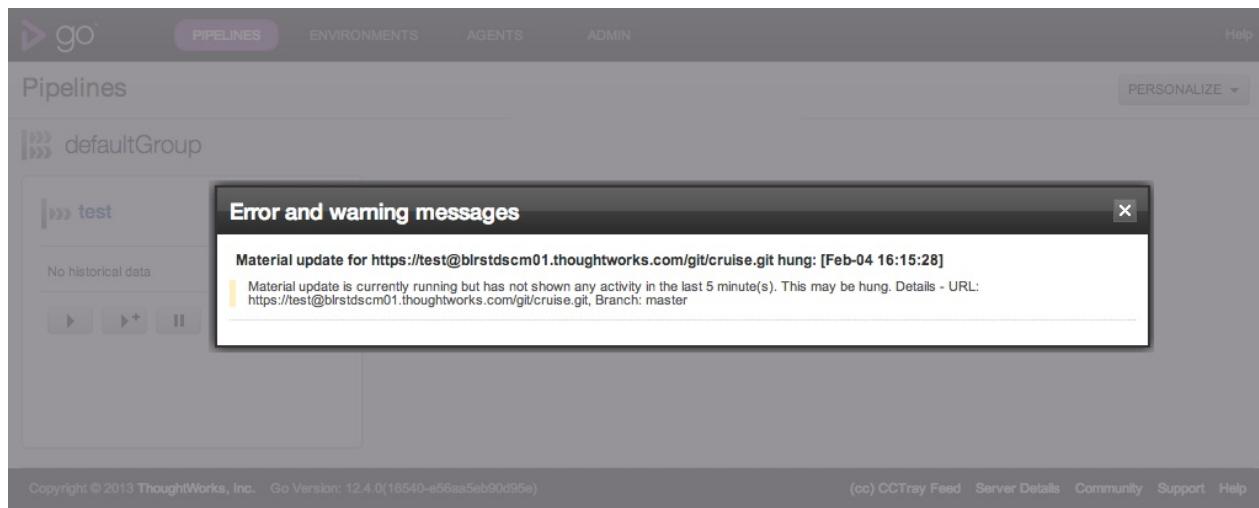
Go server polls for changes to all materials of 'Auto Triggered' pipelines. By default, polling occurs every minute and ten materials at a time. The polling interval and the number of materials to be polled simultaneously are configurable.

Go uses SCM commands to poll for changes. For example, to check for any new changes in SVN repository the following command is used:

```
svn log --non-interactive --xml -v -r HEAD:'revision' 'repository-URL'
```

The SCM command used by Go server can hang with no output. Invalid configuration, network issues, console input block are some of the causes for such a situation. Such scenarios cause pipeline scheduling delays and also lead to performance degradation

Such a scenario is notified to the users by a warning in Server Health; when clicked shows a message similar to the one below.



## What can I do with the information

When you see warning messages like the one above

- Identify the processes running as children of Go Server
- Determine the process that's hung. The extra information like URL: "[https://test@bitbucket.org/test/git\\_repo.git](https://test@bitbucket.org/test/git_repo.git)" in the warning information should help you with this.

- On linux system, you should see lines like these:
- Kill the process and all its children (the whole process tree).
- Ensure that the warning message goes away from Server Health.

## Configuring warning time

---

- Go server waits for 15 minutes (of no output) before it warns user about possible hung material update. User can modify this wait time using a System Property: 'material.update.inactive.timeout'.
- On linux installations of Go server, add the following line to /etc/default/go-server.

```
export GO_SERVER_SYSTEM_PROPERTIES='-Dmaterial.update.inactive.timeout=20'
```

The above configuration sets the time that Go server uses to determine if a material update is possibly hung, to 20 minutes.

- On Windows, add the following line in the *wrapper-properties.conf* file in the config folder of the Go server installation where x is 1 more than the highest number in *wrapper-server.conf* and *wrapper-properties.conf* combined.

```
wrapper.java.additional.x='-Dmaterial.update.inactive.timeout=20'
```

The above configuration sets the time that Go server uses to determine if a material update is possibly hung, to 20 minutes.

# Artifact integrity verification

## Overview

Go verifies artifact integrity to ensure that they are unchanged from the point of origin. While executing a job, Go applies the following rules if the checksum of the downloaded artifact does not match the checksum at the time of generation of the artifact.

- If the artifact was uploaded using the artifact API, a warning is displayed in the console output for the job
- If the downloaded artifact is different from the point of generation, the job will be failed with an error in the console output for the job.
- If Go is unable to fetch the original checksum for the downloaded artifact, a warning is displayed in the console output for the job.

Users who download artifacts for a job from the artifacts tab on the dashboard can verify their integrity by using the md5.checksum file within the cruise-output folder available on same tab. The file contains the name and checksum for each artifact saved by the job.



## Also see...

- [Managing artifacts and reports](#)
- [Auto purge old artifacts](#)

# Notifications

The Notifications page is to customize the email notifications for the current logged in user.

The screenshot shows the 'go' application interface with a dark header bar containing the logo and navigation links: PIPELINES, ENVIRONMENTS, AGENTS, and ADMIN. Below the header is a light gray 'Preferences' section. Under 'Preferences', there are two tabs: 'Notifications' (which is selected) and 'OAuth Access Tokens'. The main content area is titled 'My Notifications' and contains two sections: 'EMAIL SETTINGS' and 'FILTERS'.

**EMAIL SETTINGS**

- Email: email@email.com
- Enable email notification

**My check-in aliases**

No matchers defined

**FILTERS**

Pipeline: cruise, Stage: dev, Event: All

Only if it contains my check-ins

**Notification Filters**

Pipeline	Stage	Event	Check-ins Matcher
go-release-latest	dev	All	Mine

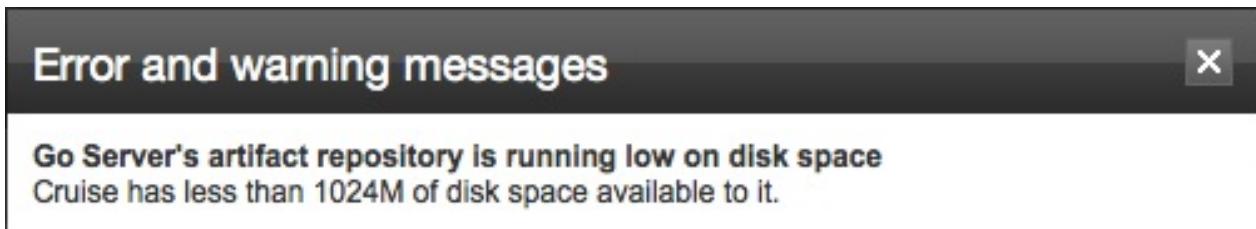
**Buttons:** ADD, RESET, DELETE

## Key

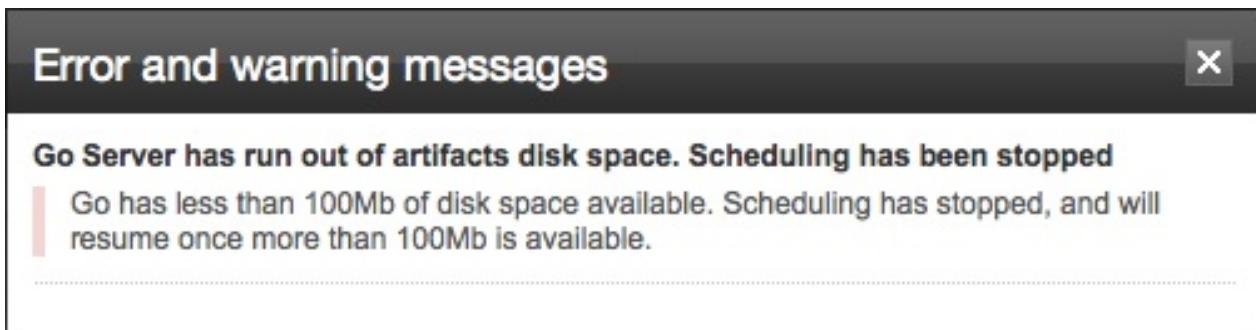
1. The email address to which email notification from Go will be sent.
2. Tick to enable email notification for the current logged in user.
3. 'My check-in aliases' is a comma separated list of aliases used for check in.

# Running out of disk space

After you've had Go running for a while, you may notice the following warning box when browsing Go:



If you don't do anything about it, you'll end up seeing the following error:



Go will stop scheduling new pipelines until you make more room, either by compressing large files, attaching a larger hard drive, or by deleting unused artifacts. You could also let Go manage artifact disk space by enabling auto purge of old artifacts.

## Auto delete artifacts

### Introduction

Go can be configured to automatically delete artifacts if the available disk space on the server is low. Go will purge artifacts when available disk space is lower than the given value. Artifacts will be purged upto the point when available disk space is greater than a defined value.

### Configuration

#### Specify artifact purge start and end limits

You must be logged in as an admin user to configure this step.

1. Navigate to the Admin section on the Go dashboard.
2. Navigate to the Pipeline Management sub-section
3. Specify when Go should begin to purge artifacts in the first edit box.
4. Specify when Go should stop purging artifacts in the second edit box.

## Pipeline Management

Artifacts Directory Location  
artifacts

Auto delete old artifacts:

Never

When available disk space is less than  GB, and keep deleting until  GB is available

Default Job Timeout

Never timeout

Cancel after  minute(s) of inactivity

**SAVE**   **RESET**

## Never delete artifacts for a stage

You must be logged in as an admin user to configure this step.

You can disallow deletion of artifacts from a particular stage so that those artifacts are excluded during deletion. This option can be set in the stage editor for a pipeline. This option can be set for stages that are important so that artifacts for the stage are preserved.

1. Navigate to the admin section on the Go dashboard.
2. Navigate to the pipelines section and choose a pipeline to edit
3. Navigate to the stage settings for the stage

The screenshot shows a sidebar with a tree view of a project named 'my\_app' containing stages like 'defaultStage', 'compile', 'test2', and 'dist'. The 'defaultStage' is selected. The main area is titled 'my\_app » defaultStage' and contains tabs for 'Stage Settings', 'Jobs', 'Environment Variables', and 'Permissions'. The 'Stage Settings' tab is active. It shows a 'Stage Name\*' input field with 'defaultStage'. Below it, 'Stage Type:' has a radio button for 'On Success' selected. There are three checkboxes: 'Fetch Materials' (checked), 'Never Cleanup Artifacts' (checked and highlighted with a red arrow), and 'Clean Working Directory' (unchecked). A note at the bottom says '\* indicates a required field'. At the bottom are 'SAVE' and 'RESET' buttons.

1. Check the box 'Never Cleanup Aartifacts'

## Also see...

- [Managing artifacts and reports](#)
- [Clean up after cancelling a task](#)

## Compress large log files

In many cases, the easiest thing to do is compress some of the larger artifacts that you won't frequently have need for. For example, if you have a large log file named 'test.log' and you're running Go server on a unix machine, the following script will gzip those files that haven't been modified in the last 10 days

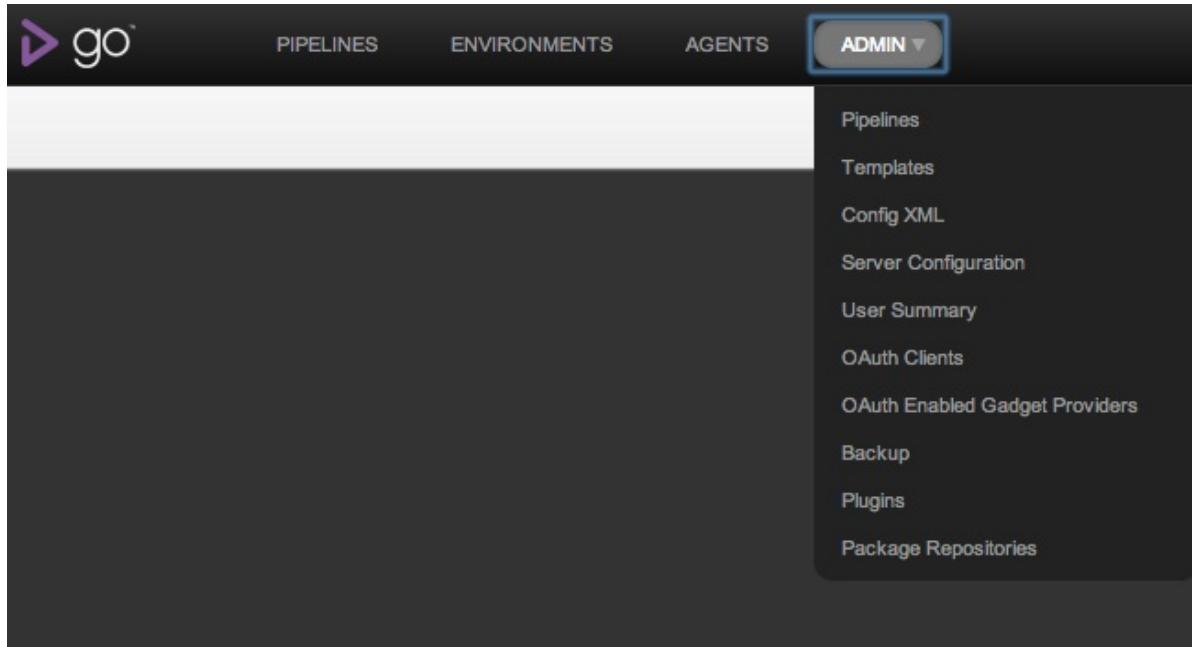
```
find /var/lib/go-server/logs/pipelines -name test.log -mtime +10 -type f -exec gzip -
```

Now, if you add this to a system [crontab](#), your server can compress large artifacts automatically.

# Move the artifact repository to a new (larger) drive

If compressing large artifacts is not giving you enough free space, another thing you can do is attach a larger disk drive to store artifacts. After the drive is attached to the system, we can easily change the location Go uses for it's artifact repository.

- Find the location of the Go configuration file
- Navigate to the [Admin](#) section



- Click on the "Config XML" tab
- The location of the configuration file is listed here



- Install the new drive
- Shut down Go server
- Copy all files from the original artifact repository location to the new drive
- Change the artifact repository location in the configuration file

```
<server artifactsDir="/path/to/new/artifacts">
...
</server>
```

- Start up Go server and verify you still have access to old artifacts

# Delete unused artifacts

---

Another option for making more room is to remove unused (or easily recreatable) artifacts. You may also have old pipelines that you no longer need.

The directory structure of the artifact repository makes selecting which artifacts are safe to delete easier. The format is:

```
[artifacts-dir]/pipelines/[pipelineName]/[pipelineLabel]/[stageName]/[stageCounter]/
```

Keep in mind that there are two files that Go needs in order to display the [Job](#) or [Stage](#) details pages

- cruise-output/console.log
- cruise-output/log.xml

# **Beta**

---

## **Features**

---

# Beta feature: Comment on a pipeline run

Note: This is a beta feature, which is turned off by default in Go 14.4.0. It can be turned on by using the feature toggle API, for this feature. If you are using curl, this is what you will need to do:

```
curl -d 'toggle_value=on'  
'http://go_server/go/api/admin/feature_toggles/pipeline_comment_feature_toggle_key'
```

When authentication is turned on in your Go Server setup, add the --user option to the curl command, like this:

```
curl --user username:password -d 'toggle_value=on' ...
```

Each pipeline in the [pipeline activity](#) page can now be annotated with a comment. This text can be seen by all other users who can access this page, for those pipelines.

When this feature is turned on, the pipeline activity page looks like this:

The screenshot shows the Go Server interface with the following details:

- Header:** go, PIPELINES, ENVIRONMENTS, AGENTS, ADMIN ▾
- Pipeline Runs:** There are four pipeline runs listed, each with a number, revision, timestamp, trigger information, and a progress bar.
- Run 4:** Number 4, revision 566274c95..., triggered 9 minutes ago by anonymous. Progress: stage1 (green), stage2 (green), stage3 (green). **ADD COMMENT** button.
- Run 3:** Number 3, revision 566274c95..., triggered 9 minutes ago by anonymous. Progress: stage1 (green), stage2 (green), stage3 (red). **ADD COMMENT** button.
- Run 2:** Number 2, revision 566274c95..., triggered 9 minutes ago by anonymous. Progress: stage1 (green), stage2 (green), stage3 (red). **ADD COMMENT** button.
- Run 1:** Number 1, revision 566274c95..., triggered 9 minutes ago by anonymous. Progress: stage1 (red), stage2 (grey), stage3 (grey). **ADD COMMENT** button.

Clicking on the "Add Comment" button for a pipeline run brings up a text box for you to enter some text. It looks like this:

go

PIPELINES ENVIRONMENTS AGENTS ADMIN ▾

Test PAUSE

stage1 stage2

**4**  
revision: 566274c95...  
9 minutes ago  
Triggered by anonymous

**3**  
revision: 566274c95...  
9 minutes ago  
Triggered by anonymous

**2**  
revision: 566274c95...  
9 minutes ago  
Triggered by anonymous

**1**  
revision: 566274c95...  
9 minutes ago  
Triggered by anonymous

**Comment on pipeline: Test label: 3**

I know why this failed! I can make it succeed!

SUBMIT

This screenshot shows a Go CI pipeline interface. At the top, there are navigation links for PIPELINES, ENVIRONMENTS, AGENTS, and ADMIN. Below that, a pipeline named 'Test' is shown with a 'PAUSE' button. The pipeline has two stages: 'stage1' and 'stage2'. There are four runs listed: Run 4 (revision 566274c95..., 9 minutes ago, triggered by anonymous), Run 3 (revision 566274c95..., 9 minutes ago, triggered by anonymous), Run 2 (revision 566274c95..., 9 minutes ago, triggered by anonymous), and Run 1 (revision 566274c95..., 9 minutes ago, triggered by anonymous). A modal window titled 'Comment on pipeline: Test label: 3' is open, containing the text 'I know why this failed! I can make it succeed!' and a 'SUBMIT' button.

Once you click "Submit", after entering some text, that text shows up against the pipeline run, like so:

Test PAUSE

stage1 stage2 stage3

**4**  
revision: 566274c95...  
9 minutes ago  
Triggered by anonymous

**3**  
revision: 566274c95...  
9 minutes ago  
Triggered by anonymous

**2**  
revision: 566274c95...  
9 minutes ago  
Triggered by anonymous

**1**  
revision: 566274c95...  
9 minutes ago  
Triggered by anonymous

There we go!

EDIT COMMENT

I know why this failed! I can make it succeed.

EDIT COMMENT

ADD COMMENT

ADD COMMENT

This screenshot shows the same Go CI pipeline interface as before, but now the comment has been submitted. The 'comment' row for Run 4 now contains the text 'There we go!' and a 'EDIT COMMENT' button. The 'comment' row for Run 3 now contains the text 'I know why this failed! I can make it succeed.' and a 'EDIT COMMENT' button. The 'ADD COMMENT' buttons for the other runs are still present.

# **RELEASE**

---

# **HISTORY**

---

# What's new in Go

---

Go has changed its release naming convention from the previous practice of major.minor.bugfix nomenclature. The major version will now be the year of release (YY). The minor version n will indicate the nth release for the year. For ex: 13.2 will be the second release in 2013.

Since release 14.2, the release notes for Go are available on the [go.cd website](#).

## Go 14.1

---

### Features

- Added capability to create a [Task plugin](#).

### Bug fixes

- Fixed a corner case issue around job reschedule. ([link](#))
- Fixed issue where SVN post-commit hook was not passing along credentials. ([link](#))
- Fixed issue with adding users via UI with a . (dot) in the username. ([link](#))
- Go Agent handles UnknownHostException by substituting host name with a generated name. ([link](#))
- Task running executables in the working directory should now be prefixed with ./ ([link](#))

## Go 13.4.1

---

### Enhancements

- Licence restrictions of agents and users have been removed.

### Bug fixes

- UI performance improvement (inlining Rails partials).

# Go 13.4

---

## Features

- [Template Admin](#): Users in Go can now view and edit a template for which they have permissions.
- [Pipeline Search](#): Allows a user to search for a specific pipeline on the pipeline dashboard.

## Enhancements

- [Pipeline Edit](#): Go Administrators and Pipeline Group Administrators can now navigate to edit a pipeline through a single click on the dashboard.
- Performance improvements have been made for pipeline dashboard page and preferences page of Go.

## Bug fixes

- Parsing of NUnit (Version 2.6) XML report.

# Go 13.3

---

## Features

- [External Package Repositories](#): Go supports external packages repositories as materials, and changes to packages in these repositories can trigger Go pipelines.

## Enhancements

- RPM installer can optionally defer starting of the Go Server upon upgrade. Setting the environment variable DO\_NOT\_START\_SERVICE=Y will defer starting Go Server upon installation/upgrade.
- Post 13.2.1, further performance improvements have been incorporated with regards to Go Server startup time.
- UI changes to move user related actions under a menu item which will appear when a user clicks on his/her username on the header.

## Others

- As per our earlier [deprecation notice](#), Go has removed support for TF CLC and Visual Studio clients when using TFS.

## Go 13.2.2

---

### Bug fixes

- Fixed bug in SVN Post-Commit hook implementation to handle commits in quick succession
- Microsoft TFS SDK downgrade - the TFS SDK was upgraded from v10.1 to v11.0 in Go 13.2.1 but a few customers reported that their TFS material updates started failing intermittently. It looks like there was a [memory leak in TFS SDK v11.0](#), therefore we are reverting to TFS SDK v10.1 with this minor release of Go to take the time and investigate the issue further.

## Go 13.2.1

---

### Enhancements

- [Template View](#): All Pipeline Group Admins and Go Administrators can now view templates while creating or editing pipelines.
- [User Delete API](#): A new API using which you can delete a disabled user.
- Check-in comments in [Value Stream Map](#): Showing more details, like the check-in comment, date and author for every Source Control Material in the Value Stream Map.
- [Improving Server Startup Time](#): The database queries that run when the server starts, have been optimized. Also, additional JVM properties have been provided to delay material polling and pipeline scheduling, thereby, improving start up time.

### Bug fixes

- Perforce material polling when a changeset with no commit message is submitted.
- The 'DO NOT SHOW ME AGAIN' button functionality in the License Expiry Warning popup.
- Showing timestamp tooltip in a friendly format on the Environments page.
- Removing duplicate links in pipeline stages atom feed.
- Showing appropriate agents as disabled when adding agents to an environment.
- Supporting java7-runtime-headless for Go debian installer.

- Sending email only once when notifications are configured on a stage for multiple events.
- Removing deprecated environment variables set for a job like CRUISE\_PIPELINE\_NAME, CRUISE\_PIPELINE\_COUNTER etc.
- Removing 'Add New Property' feature from job details page.

## Go 13.2

---

### New features

- **Value Stream Map**: Visualize end-to-end pipeline dependency. See what revisions triggered a pipeline and which dependent pipelines were triggered by it. Track changes from check-in to deploy.
- **Better support for concurrent edits to configuration**
  - Config Merge : Multiple users' changes to configuration are now merged by Go.
  - Split Pane : A user-friendly interface appears in the config xml tab in case of a merge conflict. It displays the latest config as well as the user's changes, so that a Go administrator can edit config without losing any changes!
- **Support for named branches in Mercurial**
- **Config Diff**: Go administrator can now view configuration changes between two stage runs.
- **Delete disabled agents from agents tab**.

### Enhancements

- Go now allows a timer triggered pipeline only when there are new changes.
- Better usability in stage and job history widgets. Information about config change on stage detail page.
- **Clobber option is now used in perforce during checkout.**
- Ability to specify if a pipeline is manual or automatic from pipeline options page.
- A file called wrapper-properties.conf can now be created in windows installations to hold custom properties. This can be done on both **server** and **agent** to avoid overwriting custom properties during upgrade.

### Bug fixes

- Case sensitivity issue with pipeline name in pipeline material.
- Temporary agent launcher files do not get deleted when Windows machine is shutdown.

# Go 13.1

---

## New features

- **Command Repository** : How do I run maven with Go? Is it possible to do an EC2 deploy with Go? What is the syntax if I need to execute a remote command on a linux box? The answer to these and more are now provided within Go.

## Enhancements

- **Notification if material update is hung** : Go server becomes less responsive when some of the processes that it invokes to do material updates stop responding. Now you get to know when this happens and take suitable steps.
- Support for multiple organizational units (OUs) in [LDAP configuration](#). This allows finer-grained access control. If your organization has multiple OUs in your corporate LDAP, you can now choose to specify those OUs, whose users are allowed to use Go.
- **Additional agent APIs**: Go now provides an API to list details of all agents and another API to delete disabled agents.
- Ability to search community forum from help documentation. You can now look up community articles from within the help documentation by clicking the Help icon on the center right of each help page.

## UI changes

- Docking of primary header
- Docking of breadcrumbs

## Bug fixes

- When a "run on all agents" job involves more than 100 agents, some of the jobs failed reporting completion
- Run-if conditions is shown for Cancel task
- Extraction of [pipeline templates](#) not retaining parameters
- Fan-in resolution not happening in some scenarios
- nunit test reports not parsed by Go due to case-sensitivity of file extension
- New pipeline created without using templates shows parameters of the first template, if any
- cctray breakers list returns 'Unknown' as breakers when pipeline has dependency

material

- Default column not retained for large search results in add user screen
- In environments page, icon indicating revisions available for deployment is truncated
- Check All functionality not working in Add agents tab in Environments page
- Edit dialogs in Environment tab display incorrect title
- In pipeline configuration page, when a pipeline is paused/unpaused, the same is not reflected in the screen until refresh
- In certain conditions, sorting of Agents by Free Space throws error when using OpenJDK.

## Go 12.4

---

### New features

- **Support for OpenJDK** : Oracle JDK (formerly Sun JDK) is no longer a mandatory requirement for Go server and agent. Go now needs a Java Runtime Environment (JRE) with version 6 or above - either Oracle or Open JRE.
- **Post-commit hook for Subversion** : In organizations with a large number of subversion repositories, regular polling can lead to huge network traffic. The generic post-commit hook avoids the need for regular polling. It enables appropriate Go pipelines to be triggered based on the commit to the corresponding SVN repository.

### Performance Improvements

- **Go configuration saves are faster** : Enterprise installations have large configurations, which also undergo a lot of changes. Optimizations have been done to reduce the time taken in this area.
- Better handling of cctray requests while server is starting to improve dashboard accessibility.

### Enhancements

- **Line breaks displayed in commit messages** : This improves the readability of commit messages.
- Go now deletes the associated cache as part of [auto deletion of artifacts](#).
- The username of the user who triggered a build is now available as a [job environment variable](#).
- On Windows, Go service no longer needs Admin privileges. It runs with Local

System Account.

- Official support for Google Chrome browser.
- Better error message when licensing limitation is reached.

## Bug fixes

- Minor UI issues
- Stage feed for pipeline fails when all the materials of the pipeline triggered with lower revision compared to the previous trigger
- Case sensitivity issue with pipeline name in path and config
- Unable to delete a user role if it contains dots
- Tests tab under jobs shows no results if the results were in subfolders of the test reports folder.
- Memory leak due to open password file
- Custom tabs with external contents renders incorrectly

## Go 12.3.2

---

### Bug fixes

- *Fan-In* feature corner cases.
- Failure when a Stage feed contains historical dependent pipelines that were deleted from the configuration.
- [Missing horizontal scroll on Job console page.](#)

## Performance Improvements

- Upgraded JGit library. [Details](#).

## Go 12.3.1

---

### Enhancements

- **Go server for Mac OSX - Lion and Mountain Lion** : Are you a Lion/Mountain Lion user and could not try out Go? Go Server app now works on the new Mac versions. Do remember that Mac Go server is primarily for evaluation.
- **Logging LDAP login failures** : Enterprise customers can troubleshoot LDAP errors or login failures faster, thanks to improved logging support.

- **UI improvements** : Console and custom tabs now extend to the height of the browser window. And we have a "Top" icon to quickly get you upto the top of the page.
- **Filter agents by exact search** : Now you can specify the values in quotes to do an exact search. The result - you get exactly what you want and nothing more.

## Bug fixes

- Fixed an error triggering a pipeline on the environments page

## Go 12.3

---

### New Features

- **Go's new "Fan-in" feature allows you to model your workflow to fail fast:** Continuous Delivery is about failing fast to learn fast. Early failure is cheap and easy to fix. So it really comes down to getting short feedback loops across your build-test-release workflow. What can you do with "fan-out" and "fan-in"?
  - package the right versions of your components - eliminate spurious builds
  - parallelize testing activities in different environments to find issues as quickly as possible
- **Are you happy with a pipeline and are ready to create a template? Extract a template!:** This makes scaling easier and promotes consistency in your build-test-release workflow. A common use case is when you want to start using your deployment mechanism across multiple environments.
- **Pipeline Group administrators can now view and edit the underlying XML configuration.** Some administrative tasks are easier done using XML (and believe it or not some users prefer XML!). We took this away in a previous release and are bringing it back in a better form.

### Enhancements

- **Silent (Unattended) Windows Installer.** You no longer need to stick (and click) around when you install a Windows agent. Thank you for your patience!
- Your **CCtray client** now shows you who broke the build
- **API to cancel a stage**

## Go Community Edition

- Go Community now has access to all features
- Teams of up to 10 users can now have access to a fully featured Continuous Delivery tool for free
- This includes [templates](#), [Go Environments](#), pipeline groups and pipeline group security

## Bugs Fixed

- The [Pipeline Compare](#) now uses *push date* and not *author commit date*.
- Check connection issue with Perforce

## Go 12.2.2

---

### New Features

- **TFS SDK Support:** Go Server uses [TFS Java SDK v10](#), as the default, to support TFS materials from this release. Usage of SDK makes the TFS implementation scale better. The requirement of having TFS Command line utility as a pre-requisite for TFS support is no longer necessary.
- **Go Server ID:** New [serverID](#) attribute has been added to server tag in the config XML to assist Go Server identification in deployment environments involving multiple Go servers.

### Performance Enhancements

- Memory usage during the process of test artifact analysis has been optimized. This will result in reduced memory footprint of the Go server.
- TFS CLC command and TFS Visual Studio Command implementations for modification check on the server has been optimized. No workspace and work folder mapping is being done on the server side to determine modifications in TFS repository.

## Go 12.2.1

---

Performance fixes encompassing - memory usage optimization and concurrency.

## Go 12.2

---

## New Features

- **Use Environment Variables for passwords:** Go's environment variables now have an encryption option so that you can comfortably and safely store and use passwords in Go. You and your auditors will be happy.
- **Store each artifact in one place and then fetch it whenever and wherever:** You can now fetch artifacts from any ancestor pipeline. For example, this comes in handy when you want to deploy the same binaries using the same deployment process to various environments. This ensures you test your deployment process many times all the way out to production.

## Enhancements

- **New environment variables for material revisions:** Two additional environment variables GO\_FROM\_REVISION and GO\_TO\_REVISION have been introduced during the run of every task. This is useful when a pipeline instance picks up multiple changesets from your SCM material and you have tasks that need information on which files have changed in this instance.

## Bugs fixed

- Fetch Artifact task can be used to fetch artifacts from a particular stage in an [upstream pipeline](#) or stages before it. Stages after the upstream stage **can not** be fetched from, because they may not be complete when the fetch task executes. Prior to 12.2 Go would allow fetching artifacts from stages after the upstream stage in the pipeline. For example: Let pipeline 'build' have 3 stages named 'compile', 'package' and 'test'. Let a downstream pipeline 'deploy' have a dependency on the 'package' stage of 'build'. With this dependency you can fetch artifacts from 'package' or any stage before that i.e 'compile'. From Go 12.2, you cannot fetch artifacts from the 'test' stage with this dependency. To fix this, you will have to change the dependency to the 'test' stage of 'build'. With this dependency you can fetch artifacts from any stage of the pipeline i.e 'test' or any stage before that.

This fix is not backward compatible. Hence, configurations violating this rule need to be fixed manually before upgrading to this version.

## Go 12.1

---

### New Features

- **Support for TFS SCM:** Use TFS as a material for your pipeline.
- **One-click backup:** Use the administration interface to backup Go.

## Enhancements

- **Permanently pause pipelines:** The state of a paused pipeline and comment is maintained across server restarts. Hence, you can now permanently disable pipelines without losing historical data for a pipeline.

## Go 2.4

---

### Performance Enhancements

- **Tune Go for Better Performance:** Go has been tested with a large number of pipelines, agents and load. Performance will vary based on your server configuration and usage patterns. Use this guide to optimize performance.
- **Faster Dashboard:** The [Pipelines Dashboard](#) renders much faster even with a large number of pipelines.
- **Faster Artifact Transfers:** We reduced artifact upload and download times.
- **Other:** We improved how Go manages memory.

### New Features

- **Filter on Agents page:** Use tag:value style syntax to easily find the agents you want to manage.
- **Auto registration of remote agents:** You can now auto approve remote agents without having to enable them through the agents dashboard.

## Enhancements

- **Personalize the Pipelines Dashboard:** This is now a user preference in Go.
- The online [documentation](#) now has search so you can quickly get help.
- Get the list of [scheduled jobs](#) using API.
- Configure site urls using the server configuration UI.
- Users will get a warning if the Go license is about to expire

## Bugs fixed

- GO\_P4\_CLIENT environment variable had different behaviour when destination

directory is set for perforce materials.

## Go 2.3

---

### New Features

- **Visualize build duration over time:** Visualize trends in your build's duration with a brand new graphical chart
- **Clone a pipeline:** Use the admin UI to clone an existing pipeline.
- **3 Step Pipeline Wizard:** Use the new pipeline wizard to create a pipeline in 3 easy steps.
- **Job history on an agent:** You can view the history of all the work done by an agent. Analyze problems in your execution environment to identify flaky jobs on particular agents.
- **Trace a pipeline instance back to its configuration:** Go now maintains a history of all changes made to its configuration allowing you to audit all configuration changes. You can view the Go configuration for each run in the Config tab for a stage.
- **Job Timeout:** Non-responsive jobs are now detected by Go and can be configured to automatically cancel after a timeout period.
- **Auto purge old artifacts:** Configure Go to manage server disk space by automatically purging old artifacts.
- **Artifact integrity verification:** Go verifies every file retrieved from its repository to ensure it hasn't been tampered with.

### Enhancements

- Perforce users can now make use of the GO\_P4\_CLIENT environment variable which indicates the perforce client name.
- Configure and manage custom tabs for your jobs using the admin UI.
- You can now browse the history to select a pipeline on the [Pipeline Compare screen](#).
- The Stages XML feed now includes additional details on the material check-in that triggered the pipeline and information on related mingle cards.
- Usernames and roles are case insensitive. Existing roles or usernames which have multiple entries that differ only in case will be replaced by a single instance.

## Go 2.2

---

## New Features

- **Pipeline Compare:** Go now gives you the ability to select any two pipeline instances and see exactly what changed between the two builds. The comparison will display a list of checkins grouped by material. If you use Mingle 3.3 (or greater) to manage your project, you can [associate a Mingle project](#) to a pipeline to see a list of cards that were worked on in the compare range.
- **Pipeline Administration:** A brand new UI for administering pipelines which eliminates the need to configure via XML.
- **Password Encryption:** All passwords stored in Go's configuration file are now encrypted for increased security. In prior versions, although passwords were only accessible by system administrators, they were visible in plain text. This is no longer the case.
- **Version Controlled Configuration:** Go now maintains a history of all changes made to its configuration allowing you to audit all configuration changes.
- **Job Re-runs:** Go now allows you to re-run specific jobs like it was possible to re-run stages. This helps in cases like environment problems where you had to rerun an entire stage. Job re-run gets you to the same state with lower agent utilization and potentially sooner.

## Enhancements

- Go Community no longer requires a license key.
- Grant and remove administrative privilege for users from the "User Summary" screen.
- The "Trigger with Options" popup now lets you search by a material pipeline's label.

## Go 2.1

---

### New Features

- The ability to authorize people to [administer the configuration of pipeline groups](#). This is especially useful if multiple projects/teams share a single Go server - it means boxes can be centrally managed, but teams are in control of their build configuration.
- Go Pipeline Gadget with OAuth 2.0 so that you can embed a view of a pipeline in any compliant OpenSocial container.
- Pipeline level Atom feeds
- A GUI for creating and configuring environments

- A GUI for server and user management
- Manage Go users from the administration UI
- The ability to [parameterize](#) templates.
- An option to force agents to perform a clean build, and to bypass version control operations.

## Enhancements

- Ensuring that when check-ins affect multiple pipelines, changes propagate through the pipeline dependency graph correctly.
- [Authorization to trigger, cancel and re-run automatically triggered stages](#)
- Pipeline locking is no longer turned on by default within environments.
- Better handling of [cloned agent](#) conflicts
- Go now displays when a stage was triggered on the [stage details page](#)
- Test failure information is now included in notification emails.
- Access to build cause information from within the [Failed Test History](#) report.
- Access to failure message and stack trace for a failing test from within the [Failed Build History](#) report.

## Go 2.0

---

### New features

### Environments

- [Create, manage, and use an environment](#)
- [Associate agents with an environment](#)

### Other Features

- [Ensure only one instance of a pipeline can run at the same time](#)
- [Set variables on an environment](#)
- [Set variables on a Job](#)
- [Run a job on all agents that match environment and resources](#)
- [Schedule pipelines based on a timer \(e.g. for nightly builds\).](#)
- [Default to kill any running tasks when a stage is canceled](#)
- [Pipeline templates help to remove duplication in the config file](#)
- [Deploy specific revisions of materials to an environment](#)

## API

- Agent API - Enable and disable an agent
- Scheduling a pipeline with specific material revisions
- Releasing a pipeline lock

## Changes

### Agents

- Bulk edit of agent resources
- Bulk enable and disable agents

### Scheduling

- Go now recognizes when you use the same source control repository in multiple pipelines, and only polls it once per minute, regardless of how many times it's referenced.
- In previous versions Go would not poll repositories in a manual pipeline. Go will now poll all repositories by default. You can prevent automatic updating of a material by setting autoUpdate="false" on the material.

### Environment variables

- CRUISE\_XXX variables are deprecated since Go 2.0. Please use GO\_XXX instead of CRUISE\_XXX (For example: GO\_SERVER\_URL instead of CRUISE\_SERVER\_URL).

## Cruise 1.3.2

---

### New features

- Go watches the disk space available for database and shows a warning message when it becomes low.
- Go shows the disk space available for pipelines folder under agent installation root.
- Pipeline label can now be customized by material revisions number.
- Link directly from pipeline history page to the different stage histories.

## Changes

- Go uses pipeline counter instead of label in [artifacts](#) and [properties](#) Restful urls.
- Mercurial material no longer updates the working copy on the server when checking for modifications.
- Go now schedules pipelines concurrently.
- Go will use the first changed material as build cause.

## Fixed

- Go no longer always executes 'hg clone' when the url ends with /.

## Known issues

- Pipelines with Mercurial or Git materials may initially take a long time to schedule. This is due to the command 'clone' running with high CPU usage.

# Cruise 1.3.1

---

## Changes

- Upgraded jetty to 6.1.18 and turned off jetty web log as default.
- Upgraded H2DB to 1.1.115.
- Improved performance and configurability Agent/Server communication. Go now supports >100 agents.
- Improved performance for configurations with a large number of pipelines and materials defined.
- Go will always use the latest revision for filtered materials once scheduled.

## Fixed

- Fixed the concurrency issue with a large number of pipelines on Current Activity Page.
- Fixed installers on Mac in terms of upgraded package of Mac in June.
- Fixed that Go did not pick up username and password for Subversion repository in some cases.
- Fixed invalid CRC error which might fail artifact uploading.
- Fixed issue that pipeline with multiple materials might not be able to schedule build.
- Fixed issue where agent console logger threads are not always stopped.

# Cruise 1.3

---

---

## New features

- Go watch artifacts disk space and show warning message when it becomes low.
- [User tab to manage email notifications, with the ability to match check-ins to users](#)
- Run multiple Go agents on one machine
- Cancelling a stage can run a process to clean up on the agent
- [Conditional task execution for jobs](#)
- Support for [Git branches and submodules](#)
- Go server warns you when its disk space is running low
- [Set environment variables containing the source control revision id when running a task](#)
- We now support SUSE [[serveragent](#)] (tested on Linux Enterprise Server 10 SP2).
- We now support Solaris [[serveragent](#)] (tested on SunOS Solaris10 5.10).

In the professional edition:

- Ability to group (and secure) similar pipelines
- Pipeline dependencies
- Lock down who can view and operate pipelines
- More than one material per pipeline

## Changes

- Changes with multiple materials highlights what has actually changed since the previous revision. If you used multiple materials (e.g. more than one source control) in the same pipeline, Go shows the latest revision for each of the materials, even if that is the same as the previous pipeline instance. Go now highlights the changes since the previous pipeline instance.
- You can now define groups of pipelines. This makes it easier to configure and define security on pipelines based on logical grouping for your organization. See [change permissions for different actions](#) for more information.
- Go server can now be downloaded as a single jar. For more information see [running Go without an install](#).
- The "Pipelines" tab has been removed. The Pipeline Activity page provides a much better visualization.
- Manual pipelines no longer poll source control. A pipeline with a manual first stage will not check source control until it is manually triggered. This helps reduce the number of requests to source control systems.
- Artifacts to be uploaded can now be specified with wildcards. For more information

see [uploading test reports](#).

- Artifacts are now stored on the server in directories named for the pipelines. The directory for a pipeline is "pipelineName/pipelineLabel/stageName/stageCounter/jobName". Artifacts already created will still be accessible in the existing directories, but new pipelines will use the new directory layout. This was done since some operating systems restrict the number of directory entries which can cause errors when building many pipelines.
- Go will now obscure show source control passwords. Go will show passwords as "\*\*\*\*\*" in the UI and in logs. Note that only the logs created by Go itself are obscured. If a tool or script prints out a password then it may appear in the logs.
- Forcing a pipeline will create a new pipeline instance even if there are no changes in the materials. In Cruise 1.2 forcing a pipeline would re-run the existing pipeline if there were no changes in the materials. After feedback from our users we have removed this change. If you do want to re-run an existing pipeline instance you can still do so, by clicking the re-run button on the first stage of that pipeline.
- Upgraded jetty to 6.1.18 with better performance.
- Use the nio selector from jetty to improve SSL performance.
- Licensing has been changed. You will need a new license if you have a paid version. Please go [here](#) to see details.

## Cruise 1.2.1

---

### Changes

- Fixed the issue of full-checkout on agent side for Perforce.
- Fixed the issue of full-checkout on agent side for when the option of 'checkexternals' is on.
- Manually triggering a pipeline will generate a new pipeline instance even though there is no modification detected. Also it will update to most recent revision, including filtered files.

## Cruise 1.2

---

### New features

Cruise 1.2 offers many new features and enhancements in terms of usability and performance.

## Support tracking tools to integrate with other management tools.

Go now supports linking to a story / bug tracking tool like Mingle from commit messages in Go. Use the following syntax:

```
<pipeline name="cruise">
  <trackingtool link="http://mingle/projects/cruise/cards/${ID}" regex="#(\d+)" />
  ...
</pipeline>
```

**Description:** Using the regular expression above, if your commit messages contain messages of the form '#2350', you could see the link wherever you can see the commit message.



## Set properties from artifact using XPath automatically.

After a job has finished on an agent, Go can parse any XML files on the agent and set properties on the server. You can use this to harvest metrics like code coverage or cyclomatic complexity from your reports into Go, and then export the complete history of your properties from the job detail page to a spreadsheet for custom reporting and analysis. Use the following syntax to tell Go the XML files to parse, the XPath to get the property value, and the property name. The following example extracts useful coverage metrics from EMMA reports..

```
<job name="test_analysis">
  <properties>
    <property name="coverage.class" src="target/emma/coverage.xml" xpath="substring-before(coverage/coverage-class, '#')"/>
    <property name="coverage.method" src="target/emma/coverage.xml" xpath="substring-before(coverage/coverage-method, '#')"/>
    <property name="coverage.block" src="target/emma/coverage.xml" xpath="substring-before(coverage/coverage-block, '#')"/>
```

```

<property name="coverage.line" src="target/emma/coverage.xml" xpath="substring-before(coverage/line, '#')"/>
</properties>
...
</job>

```

**Description:** When you specify this configuration, you can see the properties in the sub-tab 'properties' on job detail page, and export their history to a spreadsheet.

Property name	Property value
coverage.block	67
coverage.class	77
coverage.line	66
coverage.method	65

## Improved CCTray feed

- The performance of CCTray feed is much better than before.
- When the stages or jobs are building, Go now supports showing the results of the last build.
- The CCTray feed now supports basic authorization. Specify your username and password in the url of cctray feed like this: [http://\[your-username\]:\[your-password\]@yourgoserver:8153/cruise/cctray.xml](http://[your-username]:[your-password]@yourgoserver:8153/cruise/cctray.xml)

**Notes:** some special characters do not work, e.g. '/'.

## Support to access re-run stages.

Although Go has always stored results of re-run stages, you can now access previous runs of a stage easily. We've introduced a 'stage counter' which indexes the runs of the stage.

- The stage history is ordered by time rather than pipeline label in the Stage Detail Page.
- Access any stage details page by the Go API  
[http://\[your\\_cruise\\_server\]:8153/cruise/pipelines/\[pipeline\\_name\]/\[pipeline\\_label\]/\[stage\\_label\]/\[stage\\_counter\]](http://[your_cruise_server]:8153/cruise/pipelines/[pipeline_name]/[pipeline_label]/[stage_label]/[stage_counter])

[stage\\\_name\]/\[stage\\\_counter\]](#)



## Support multiple repositories for Subversion

Go lets you define multiple repositories in `cruise-config.xml` if you are using Subversion as your VCS. This means you can easily create pipelines for applications that rely on multiple repositories.

```
<pipeline name="main">
  <materials>
    <svn url="http://xxx.xxx.xxx/trunk" username="yyyy" password="xxx" dest="client" />
    <svn url="http://zzz.zzz.zzz/trunk" username="yyyy" password="xxx" dest="server" />
  </materials>
</pipeline>
```

You can see material revisions for each pipeline in the sub-tab 'materials' on the job detail page and stage details page, and the tooltips as well. Note that this version of Go also treats upstream pipelines as materials, just like source control, which means you can see exactly which version of your upstream pipeline triggered this one (see the

screenshot below). Use [this syntax](#) to define dependencies between pipelines.

cruise/2.0.0.5092-19bbc302e47aa9009582b0e6e1f573908865a986/dev/1/linux-firefox-1 job | passed

SCHEDULED ON:	2010-07-01T11:17:45+05:30	COMPLETED ON:	2010-07-01T11:23:39+05:30
DURATION:	00:05:12	AGENT:	blrstdcrsuat16 (ip:10.4.8.16)
BUILD CAUSE:	modified by Jake & JJ		

Console Tests Failures Artifacts Materials Properties Console Junit-time

[Link to this tab](#)

**Mercurial trunk - https://ccepair:\*\*\*\*\*@fmtstdscm01.thoughtworks.com/cruise**

Revision: 19bbc302e47aa9009582b0e6e1f573908865a986, modified by Jake & JJ on 2010-07-01T10:07:34+05:30  
"#4198 - fixed environment page caching issue(which prevented pipeline box from moving across environments boxes on the ui)"

▲ server/webapp/WEB-INF/rails/app/views/environments/index.json.erb  
▲ server/webapp/WEB-INF/rails/spec/views/environments/index\_json\_spec.rb

Revision: c1f2ee158895fa23d85226e030aab35f58da2cf0, modified by JJ on 2010-07-01T11:10:48+05:30  
"fixing dev build"

▲ common/test/com/thoughtworks/studios/licensing/PublicKeyDecrypterTest.java

**Git twist - cruise@10.4.3.137:repo/cruise\_qa**

Revision: ef2b2e010380cc8f7de3a140ccbc55c601c2f383, modified by Shilpag & rrr <goleys@thoughtworks.com> on 2010-06-30T16:37:39+05:30  
"changing footer from cruise to go"

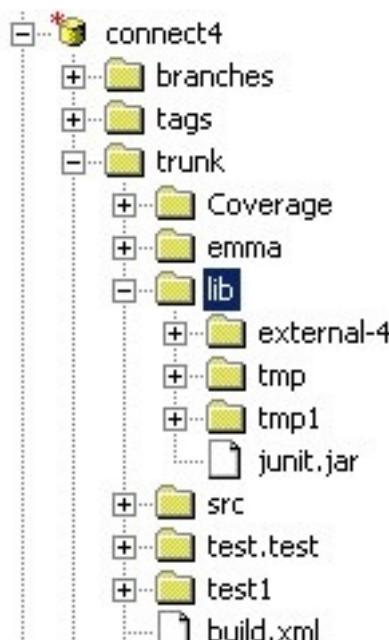
▲ cruise-twist-new/src/com/thoughtworks/cruise/page/CruisePage.java

## Ability to filter materials

Go allows you to specify files in your version control repository that shouldn't trigger a new build when they change (for example requirements documents). You can specify multiple "ignore" blocks with patterns to match, using ant-like syntax.

The pattern takes the root directory of the SCM repository as the root directory, not the sandbox in the agent side or the materials URL.

For example, the structure of a subversion repository is following:



The material and changes look like

**Subversion <http://10.18.3.171:8080/svn/connect4/trunk/lib>**

Revision: 297, modified by cce on 2009-06-13 16:17:49 +0800  
**should ignored**

 [/trunk/lib/tmp/ignored.txt](#)

If we want to ignore files under /trunk/lib/tmp, the configuration is:

```
<svn url="http://10.18.3.171:8080/svn/connect4/trunk/lib" username="cce" password="pa
    <filter>
        <ignore pattern="trunk/lib/tmp/**/*.*" />
    </filter>
</svn>
```

## Fetching artifacts

Go supports fetching artifacts from previous stage or upstream pipeline in a job by definition of \ in cruise-config.xml. Using this feature it's easy to have one job store artifacts like binaries, jars or installers, and then re-use them in a later stage. This supports best practices like only compiling your application once.[Here is an example.](#)

## Changes

- Refactored Go API and remove some calls. See the [Go API topic](#) for more details.
- Create the dependency of pipelines as the materials of the pipelines. This means you can see which version of your upstream pipeline triggered the current pipeline anywhere on the UI you see modifications. Your configuration will be upgraded automatically.
- Limit the number of materials of pipelines. Go Free Edition supports only one material, and Enterprise Edition support unlimited materials. The other versions support three materials.
- Go agent no longer bundles Ant as of version 1.2.
- Clean the folders which is not under any 'dest' folders of all material on agents.

## References

- [Installing Go server](#)
- [Installing Go agent](#)
- [Upgrading Go](#)

# Cruise 1.1

---

## New features

Cruise 1.1 offers many new features and enhancements in terms of usability and performance.

## Support for Perforce

Go now supports Perforce. In particular, we support ticket-based authentication.

## Support for Git

The support for Git on Go is similar to the Mercurial support. Git must be installed on both the server and agents.

## Role-based security

- Go now allows definition of roles composed of multiple users. You can use these roles anywhere you would restrict access by user (to control administrator access, and to control who can authorize approvals).
- As with 1.0, if security is switched off, Go allows all users anonymous read access to Go, so that anybody with access to the Go server can see the state of builds.

## Authorization for approvals

Go now allows role-based authorization for approving a stage manually. This effectively means you can make pipelines 'read-only'. You can specify individually for every stage which group of users or roles are allowed to press the approval button. Specifying roles on the first stage, if it is a manual approval, restricts who can press the "trigger pipeline" button.

## Re-run a stage

Go lets you re-run individual stages. Triggering this operation will run the stage with the same source control revisions that it was originally run with. Go automatically updates to the original revision it was run with. This is useful if you have a stage that deploys to a testing environment, and you want to re-deploy a historical build to that environment. Hover over a stage on the Pipeline Activity page or click on the re-run button on the Current Activity page. If you re-run a stage that failed and the stage passes then the rest

of the pipeline will run through as normal.

## UI improvements for pipeline activity

The Pipeline Activity tab has an improved user interface.

The screenshot shows the Go web interface with the 'PIPLINES' tab selected. A pipeline named 'deploy-go01' is displayed. Below the pipeline name is a 'PAUSE' button. The pipeline activity section lists four recent runs:

- 115**: revision: publish-rpm/..., 1 day ago, Triggered by changes. Status: In Progress (green bar).
- 114**: revision: publish-rpm/..., 9 days ago, Triggered by changes. Status: In Progress (green bar).
- 113**: revision: publish-rpm/..., 9 days ago, Triggered by changes. Status: In Progress (green bar).
- 112**: revision: publish-rpm/..., 15 days ago, Triggered by changes. Status: In Progress (green bar).

## Solaris support

With Cruise 1.1 both the server and agents are supported on Solaris.

## Better server error reporting

Go will display errors on its dashboard if the server has an invalid configuration file, if it cannot contact your revision control repository, or if the revision control client software is not installed on the server. The errors will go away automatically once the cause is resolved.

## Better console out information

Go now reports agent actions (such as checking out, or uploading artifacts) in the Go log that can be viewed through the web page. This makes it much easier to debug issues with source control systems and other Go agent actions.

The screenshot shows a tabbed interface with tabs for Console, Tests, Failures, Artifacts, Materials, and Properties. The Console tab is active. The log output in the console window is as follows:

```
[cruise] Start to prepare app-Perf/52/deploy/1/P1.S1.Job1-1ce1465d-d8ba-4433-b4fd-ac5426b1ffce on blrstdcrspbs01.thoughtworks.com [/var/lib/cruise-agent] at Mon Jun 28 15:14:12 IST 2010
[cruise] Start updating files at revision 48 from http://subversion.assembla.com/svn/tingtong/
At revision 48.
[cruise] Start to build app-Perf/52/deploy/1/P1.S1.Job1-1ce1465d-d8ba-4433-b4fd-ac5426b1ffce on blrstdcrspbs01.thoughtworks.com [/var/lib/cruise-agent] at Mon Jun 28 15:14:17 IST 2010
[cruise] Start to execute task: <exec command="ls" args="." />. Current status: passed
[cruise] setting environment variable 'GO_ENVIRONMENT_NAME' to value 'Demo-Perf'
```

## De-authorize agents through UI

The agents can be disabled from the server through the user interface which means that the agent will no longer process any work. Click the Enable button to start builds on the agent again.

The screenshot shows a table of agents managed by the Go server. The columns are Host Name, Sandbox, OS, IP Address, Status, Free Space, Resources, and Environments. The table includes the following rows:

Host Name	Sandbox	OS	IP Address	Status	Free Space	Resources	Environments
fmtdshnbgr01.thoughtworks.com			10.38.9.1	pending			no environments specified
blrstdcrsato07.thoughtworks.com	/export/home/cc/e/AutoDeployTesting/aut-agent	SunOS	10.4.8.87	lost contact	unknown	OpenSolaris   autodeploy   dist	no environments specified
fmtdshnbgr101			10.38.9.101	missing	unknown	shine   windows	no environments specified
blrstdcrsua01.thoughtworks.com	/var/lib/cruise-agent	Linux	10.4.8.1	building	9.9 GB	dev   linux   twist	no environments specified
blrstdcrsua02.thoughtworks.com	/var/lib/cruise-agent	Linux	10.4.8.2	building	9.7 GB	dev   linux   twist	no environments specified
blrstdcrsato01	C:\AutoDeployTesting\aut-agent	Windows Server 2008 R2	10.4.8.81	disabled	3.9 GB	IE-8   Windows2008   autodeploy	AutoDeploy-WindowsServer2008
blrstdcrsato02	C:\AutoDeployTesting\aut-agent	Windows 7	10.4.8.82	disabled	768.6 MB	IE-8   Windows7   autodeploy	AutoDeploy-Windows7

## Improved performance

Go now uses less memory and the web pages are significantly more responsive for very large installations with a lot of history.

## Changes

- Go no longer bundles SVN - the server and all agents need to have Subversion installed and available in the system path
- Approvals are now at the start of a stage not the end of the stage. This means that it is possible to add a pipeline that will only be run if you manually trigger it (in other words, revision control changes will not automatically trigger it). To do this, add a manual approval to the start of the first stage.

## References

- [Installing Go server](#)

- [Installing Go agent](#)
- [Upgrading Go](#)