

# General tech primer

The following is a cheat-sheet of some of the technologies used at RI.

[ [CI/CD](#) ] [ [Continuous Integration](#) ] [ [Continuous Delivery](#) ] [ [Concourse CI](#) ] [ [Terraform](#) ] [ [IDE \(Integrated development environment\)](#) ] [ [IntelliJ](#) ] [ [Atom IDE](#) ] [ [Visual Studio](#) ] [ [Linting and Static Code Analysis](#) ] [ [AWS DB services](#) ] [ [ELK](#) ] [ [Automated testing](#) ] [ [A few programming languages](#) ] [ [Serverless computing](#) ] [ [SQL vs NoSQL](#) ] [ [SQL Tables vs NoSQL Documents](#) ]

## CI/CD

This stands for Continuous Integration / Continuous Delivery. This is an automated process by which we create a 'pipeline' of software delivery, flowing from our version control to the deployment environment.

Not that this doesn't equate to continuous deployment - implementing CI/CD does not necessarily mean that we are going to be doing multiple Production deployments a day. Instead we will be deploying to Dev, and can (and often do) have a further manual step to then deploy that build to Staging, and then onto Prod. There can be (and there is) a deployment workflow and approval / CAB mechanism.

## CONTINUOUS INTEGRATION

Continuous Integration (CI) is the process of automating the build and testing of code every time a team member commits changes to version control. CI encourages developers to share their code and unit tests by merging their changes into a shared version control repository after every small task completion. Committing code triggers an automated build system to grab the latest code from the shared repository and to build, test, and validate the [shared Develop branch](#).

CI emerged as a best practice because software developers often work in isolation, and then they need to integrate their changes with the rest of the team's code base. Waiting days or weeks to integrate code creates many merge conflicts, hard to fix bugs, diverging code strategies, and duplicated efforts. CI requires the development team's code be merged to a shared version control branch continuously to avoid these problems.

CI keeps the branches clean: Teams can leverage modern version control systems such as Git to create short-lived [feature branches](#) to isolate their work. A developer submits a "pull request" when the feature is complete and, on approval of the pull request, the changes get merged into the Develop branch, and then the developer can delete the previous feature branch. Development teams repeat the process for additional work. The team can establish branch policies to ensure the master branch meets desired quality criteria.

## CONTINUOUS DELIVERY

While Continuous Integration makes sure that everyone is integrating their code with everyone else's, Continuous Delivery makes sure that code is then built, tested and deployed to the Development environment. This is usually triggered by a pull request / merge. The CI tool then builds the software and runs it through the test suite (see the automated testing section below) before finally deploying it.

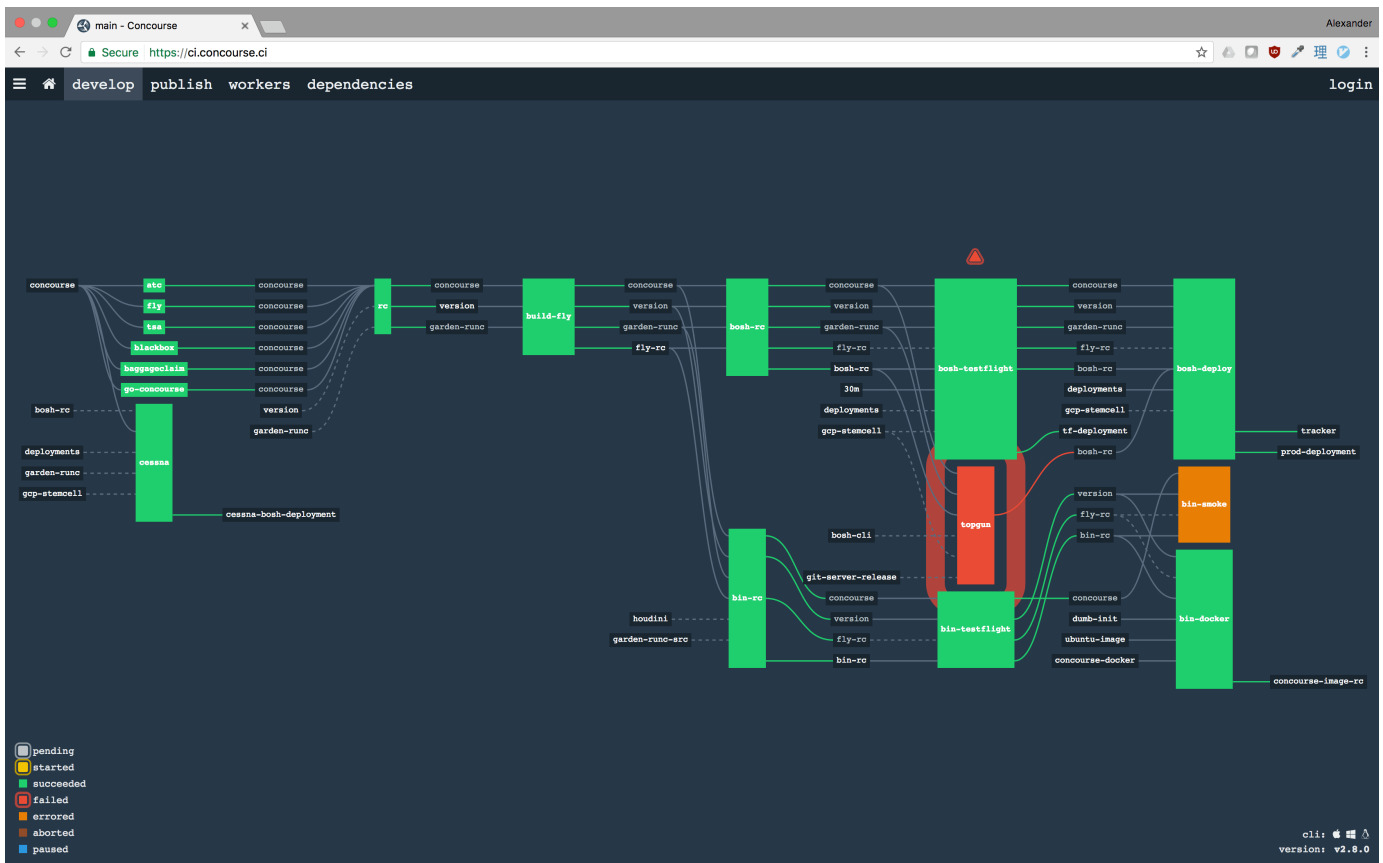
In our cloud-based world the CI/CD pipeline can go a step further, in that it not just builds and deploys the code, but does the same for the environment (infrastructure) that the code is going to be run on. This is concept of '[Infrastructure as code](#)' - that we can direct a cloud provider [such as AWS](#) to tear down an environment and recreate it multiple times a day, as part of every deployment.

Quote Martin Fowler: "Continuous Integration is a software development practice where members of a team integrate their work frequently. Each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly."

## CONCOURSE CI

This is one flavour of CI/CD software, that we have selected to be the de-facto standard at RI.

An example Concourse pipeline:



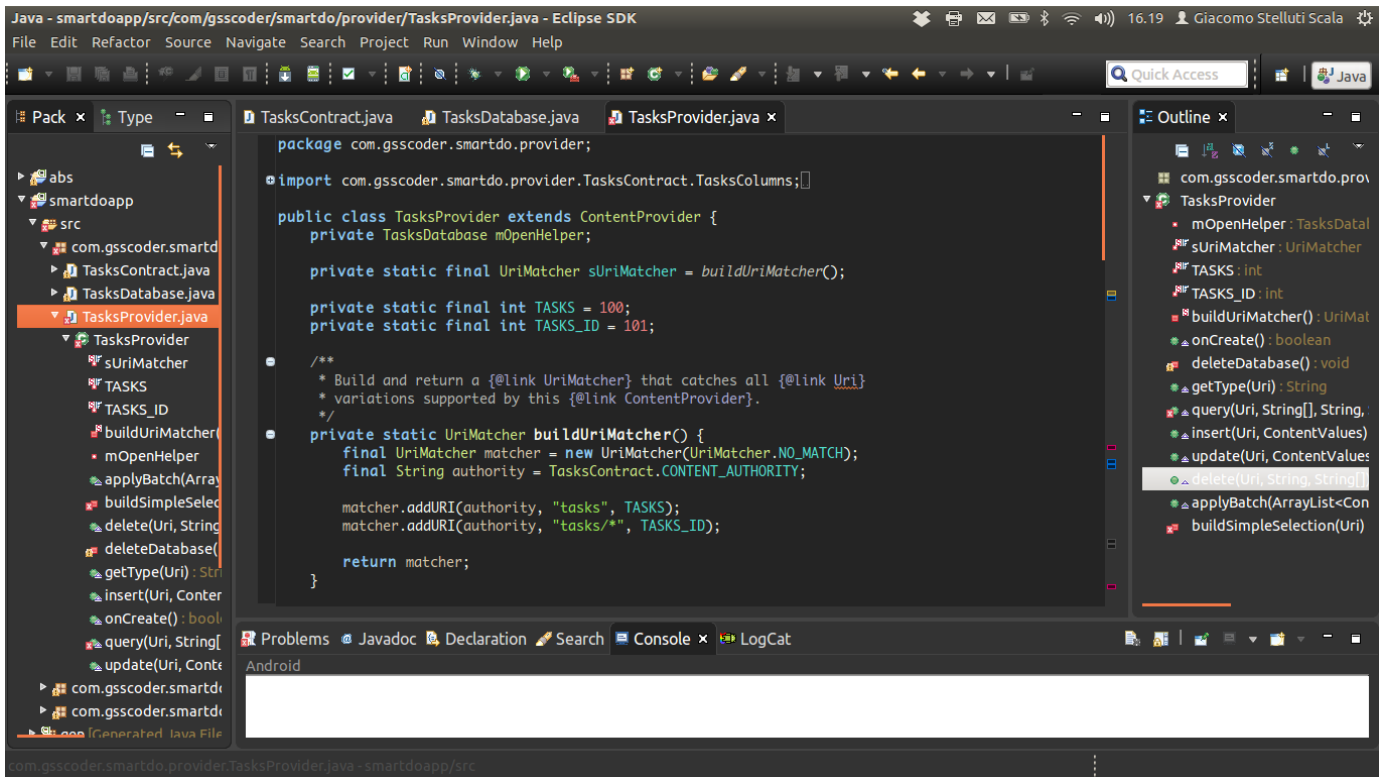
<https://concourse.ci/concepts.html>

## Terraform

I mentioned under Continuous Delivery that we can script in code what infrastructure we want for a given environment, and whenever we deploy we tear down this environment and re-create it. There are a variety of tools that enable this concept of 'Infrastructure as code', such as Ansible, Chef, Puppet and CloudFormation. The exact ins and outs as to why we as a company have decided that Terraform is the best tool for this job is outside of the scope of this page, but there's a good comparison here: <https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c>. You may also want to check out the same site's page 'An Introduction to Terraform' which gives a good hand-holding step-by-step guide to creating a simple Terraform script: <https://blog.gruntwork.io/an-introduction-to-terraform-f17df9c6d180>

## IDE (Integrated development environment)

Hopefully most developers will come across the concept of an IDE. For the non-devs amongst us, an IDE is to programming what MS Word is to desktop publishing. Imagine a software suite that consolidates the basic tools developers need to write and test software - a code editor, a **compiler** or **interpreter** and a debugger that the developer accesses through a single graphical user interface (**GUI**). An IDE will typically colour-code your code as you type it, and often will automatically indent your code for you to increase readability (though there are some programming languages where whitespace is important - yes Python, I'm giving evil looks at you, for your sins. Though frankly GoLang, you're not much better with your enforced single-line brace style). But I digress... The IDE can also offer up code hints or flag potential errors as you type, and will have an integration with your version control (Github, for example) so that you can check code out, work on it and then check it back in, all in the same application.



A typical IDE - In this case Eclipse, the IDE that I cut my teeth on.

#### INTELLIJ

IntelliJ is a type of IDE specialised for writing **Java**. While there are other IDEs that can be used for writing Java (such as Eclipse, above) IntelliJ's main selling point is it's ability to analyse your whole project and make intelligent suggestions on what you're trying to do - so the IDE makes astute debugging, autocomplete and refactoring suggestions.

#### ATOM IDE

Released by the GitHub people, Atom is a FOSS simple-as-in-straightforward text / source code editor with support for a variety of languages via plugins. The Golang plugin sorts out indentation and (required) whitespace for you, which is nice. Known for it's extendability, one of the first-class extensions available is a [collaboration tool](#) allowing for devs to work together on the same code at the same - great for remote code reviews and pair programming.

#### VISUAL STUDIO

Visual Studio is another IDE, this one from Microsoft. It used to only really be good for developing using Microsoft-related languages, notably C#, but nowadays MS have branched out a lot and Visual Studio can be used for a number of languages. That said, if you're creating a .NET application then this is the tool to use.

#### LINTING AND STATIC CODE ANALYSIS

Linting is simply the continual checking of your code for potential errors as you're typing it (i.e. before compilation); these can be syntactical errors, stylistic errors or flagging of **code smells**. Think of the red underlining you get with spelling mistakes in Outlook / Word and the like. The flag will typically inform you of what the linting application feels is the issue, and often a suggestion for an alternative - the developer can then choose whether to change or not.

#### AWS DB services

Have a read of Christina's pages here: [Which AWS DB to use and when](#) and [SQL to NoSQL Best Practices with Amazon DynamoDB](#).

Some other DB services offered in AWS:

If You Need	Consider Using	Product Type
A managed <a href="#">relational database</a> in the cloud that you can launch in minutes with a just a few clicks.	<a href="#">Amazon RDS</a>	<a href="#">Relational Database</a>
A fully managed MySQL and PostgreSQL-compatible <a href="#">relational database</a> with 5X performance and enterprise level features.	<a href="#">Amazon Aurora</a>	<a href="#">Relational Database</a>
A managed <a href="#">NoSQL database</a> that offers extremely fast performance, seamless scalability and reliability	<a href="#">Amazon DynamoDB</a>	<a href="#">NoSQL Database</a>
A fast, fully managed, petabyte-scale <a href="#">data warehouse</a> at less than a tenth the cost of traditional solutions.	<a href="#">Amazon Redshift</a>	<a href="#">Data Warehouse</a>
To deploy, operate, and scale in-memory cache based on memcached or <a href="#">Redis</a> in the cloud.	<a href="#">Amazon ElastiCache</a>	<a href="#">In-Memory Cache</a>
Help migrating your databases to AWS easily and inexpensively with minimal downtime.	<a href="#">AWS Database Migration Service</a>	<a href="#">Database Migration</a>
To build flexible cloud-native directories for organizing hierarchies of data along multiple dimensions.	<a href="#">Amazon Cloud Directory</a>	<a href="#">Directory</a>

## ELK

'ELK' stands for a collection of three open-source products — [Elasticsearch](#), [Logstash](#), and [Kibana](#) (but is also just beoming known as 'the Elastic Stack'). The trio, which was once separate, joined together to give users the ability to run log analysis on top of open sourced software that everyone can run for free.

- Elasticsearch is the search and analysis system. It is the place where your data is finally stored, from where it is fetched, and is responsible for providing all the search and analysis results.
- Logstash, which is in the front, is responsible for giving structure to your data (like parsing unstructured logs) and sending it to Elasticsearch.
- Kibana allows you to build pretty graphs and dashboards to help understand the data so you don't have to work with the raw data Elasticsearch returns.

This all comes in handy when you need time-based data analysis.

## Automated testing

Automated software testing is an alternative to manual testing where software tools, not human testers, execute pre-scripted tests on a software application: Automation tools enable testing organisations to run tests quickly and repeatedly, and so they are considered an essential component of a CI/CD pipeline.

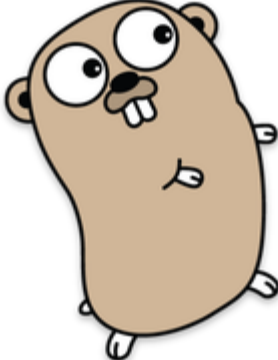
There are a number of types of automated testing, and even test-based development methodologies (see [TDD](#) and [BDD](#)), but here is just a summary of a few different test automation applications:

Application	Uses and comments
Spock	A BDD / testing framework built using the Java-related 'Groovy' language, so it can work with anything that'll run in a JVM. The test cases are written out in Groovy which requires less 'boilerplate' than Cucumber / Gherkin, meaning that this is good for devs, less good for non-technical stakeholders.
SonarQube	A <a href="#">continuous inspection</a> / <a href="#">static analysis</a> tool to manage technical debt - checks for unit tests, code duplication, quality of design, use of comments, use of coding standards, <a href="#">cyclomatic complexity</a> and potential 'general' problems.
JUnit	A <a href="#">unit testing</a> framework for Java development. Good for TDD.
Cucumber	A FOSS BDD testing tool. It used to only be for testing Ruby applications, but nowadays can be used for a variety of languages. Cucumber gave the world Gherkin - an English-based, human-readable means of describing your test scenarios that is non-technical enough that all stakeholders can understand it, but can still be utilised by the Cucumber tool. Gherkin established the now familiar 'Scenario: Given, When, Then' syntax.

Gatling	A FOSS <a href="#">performance, load and stress testing</a> application - so used for what we call 'Non-functional Testing' (i.e. testing the way that a system operates, rather than the specific behaviours of the system). Gatling is similar to the venerable Apache JMeter, but easier to set up.
Selenium	Used for testing browser-based applications, Selenium works by automating the actions the browser should take. Can also be used to automate repetitive browser-based tasks.
Ginkgo	A FOSS BDD-style Golang testing framework

### A few programming languages

There are many, many programming languages, but below are a few ones commonly used at RI. Note that I am not making a distinction between a programming language and a scripting language - this isn't something we need to go into here: [Google it if you're fussed](#).

Language	Comments
Java	<p>A general-purpose high-level OO* language. Designed to be cross-platform by running all Java applications in a <a href="#">Java Virtual Machine</a> (JVM), which also takes care of things like memory allocation. Very widely used, but has a bit of a reputation as not being the most performant language out there... This is maybe a tad unfair nowadays, but the fact is that on startup it does need the whole JVM loaded up, and so has a higher startup overhead than some other programming languages</p> <p>*Q: "Whats the object-oriented way to become wealthy?"</p> <p>A: Inheritance</p>
JavaScript	<b>Not</b> to be confused with Java - they have nothing to do with each other. (It used to be called 'LiveScript', but it got renamed to piggyback on the big Java hype around in the 90s). Javascript has traditionally been associated with client-side web, but now is getting everywhere, even on the server-side with <a href="#">Node.js</a> . JSON (JavaScript Object Notation), a very widely-used means of expressing data object in a human-readable format, comes from JavaScript.
Go	<p>AKA 'Golang' (as just 'Go' put into a search engine was considered 'unique' enough to bring back relevant results), is a language created by Google, vaguely based on C. Like Java it deals with memory allocation (<a href="#">garbage collection</a>), but unlike Java doesn't have the overhead of having to fire up a JVM in order to run, so has shorter start-up times. We tend to be opting to use Go as a default for our Microservices. It also has a cute gopher logo:</p> 
C#	Pronounced 'C sharp' like the musical note a semitone above C (one of the black keys on a keyboard) this is another OO language, this one developed by Microsoft as part of its .NET platform (and so traditionally thought of as being for Windows only, though this is not necessarily the case nowadays). Another one with garbage collection (it's quite similar to Java in style and execution) it has a very tight integration with Microsoft Visual Studio.
Bash	Standing for 'Bourne-again shell' for reasons which are not amusing enough to outline here, 'Bash' refers to both the command processor and the language used to pass commands to it; Bash is the pretty-much standard shell on Unix and Unix-like OSes (e.g Linux and MacOS). Bash commands can be (and often are) collated together into 'shell scripts' which provide a user with typically greater amount of OS integration than comes out of the box for most programming languages. A true hardcore shell scripter will write their scripts in <a href="#">Vi</a> , the greatest but also <a href="#">most user-unfriendly text editor in the world</a> .

## Serverless computing

This is a cloud computing paradigm where the infrastructure is invisibly provisioned for you, behind the scenes, as it were: You simply deploy your code and the cloud provider takes care of the infrastructure side of things for you. Needless to say that 'serverless computing' does actually still require a server to execute on, it's simply that the management of this is completely hidden from you. The advantage to this is that you don't have to spend time creating new EC2 instances, setting up the auto-scaling etc - you just deploy and it's ready. This can also save money, as you only pay for the resources used, so if your code isn't being executed then you don't pay for it. There's also an advantage in terms of code simplicity - all handlers become functions, and multithreading / parallelisation is handled for you.

## SQL vs NoSQL

Confusingly for some flavours of NoSQL DBs you can use SQL (technically SQL-like languages like [N1QL](#), but the distinction isn't something we need to go into here). The main difference between a SQL and a NoSQL DB is really down to the organisation of the data. Essentially a relational SQL DB stores its data in a structured, relational format, organised by its schema. A NoSQL DB stores its data in a non-structured, non-relational schemaless format.

Clear? Good.

A very ugly and not particularly accurate way of thinking about it is that instead of your DB > Table > Fields schema for your RDBMS, imagine a DB that has any number of tables, each with just one field - Data: You can store whatever you want in that; it essentially has no schema.

This following section lifted from <https://www.sitepoint.com/sql-vs-nosql-differences/>, as they have put the following points across much better than I could of done...

### SQL TABLES VS NOSQL DOCUMENTS

SQL databases provide a store of related data tables. For example, if you run an online book store, book information can be added to a table named `book`:

ISBN	title	author	format	price
9780992461225	JavaScript: Novice to Ninja	Darren Jones	ebook	29.00
9780994182654	Jump Start Git	Shaumik Daityari	ebook	29.00

Every row is a different book record. The design is rigid; you cannot use the same table to store different information or insert a string where a number is expected.

NoSQL databases store JSON-like field-value pair documents, e.g.

```
{
  ISBN: 9780992461225,
  title: "JavaScript: Novice to Ninja",
  author: "Darren Jones",
  format: "ebook",
  price: 29.00
}
```

Similar documents can be stored in a **collection**, which is analogous to an SQL table. However, you can store any data you like in any document; the NoSQL database won't complain. For example:

```
{
  ISBN: 9780992461225,
  title: "JavaScript: Novice to Ninja",
  author: "Darren Jones",
  year: 2014,
  format: "ebook",
  price: 29.00,
  description: "Learn JavaScript from scratch!",
  rating: "5/5",
  review: [
    { name: "A Reader", text: "The best JavaScript book I've ever read." },
    { name: "JS Expert", text: "Recommended to novice and expert developers alike." }
  ]
}
```

In a NoSQL database, data can be added anywhere, at any time. There's no need to specify a document design or even a collection up-front. For example, in MongoDB the following statement will create a new document in a new `book` collection if it's not been previously created:

```
db.book.insert(  
  ISBN: 9780994182654,  
  title: "Jump Start Git",  
  author: "Shaumik Daityari",  
  format: "ebook",  
  price: 29.00  
);
```

#### **MYTH: NoSQL supersedes SQL**

That would be like saying boats were superseded by cars because they're a newer technology. SQL and NoSQL do the same thing: store data. They take different approaches, which may help or hinder your project. Despite feeling newer and grabbing recent headlines, NoSQL is not a replacement for SQL — *it's an alternative*.

#### **MYTH: NoSQL is better / worse than SQL**

Some projects are better suited to using an SQL database. Some are better suited to NoSQL. Some could use either interchangeably.

#### **MYTH: SQL vs NoSQL is a clear distinction**

This is not necessarily true. Some SQL databases are adopting NoSQL features and vice versa. The choices are likely to become increasingly blurred, and NewSQL hybrid databases could provide some interesting options in the future.

#### **MYTH: the language/framework determines the database**

We've grown accustomed to technology stacks, such as —

- LAMP: Linux, Apache, MySQL (SQL), PHP
- MEAN: MongoDB (NoSQL), Express, Angular, Node.js
- .NET, IIS and SQL Server
- Java, Apache and Oracle.

There are practical, historical and commercial reasons why these stacks evolved — but don't presume they are rules. You can use a MongoDB NoSQL database in your PHP or .NET project. You can connect to MySQL or SQL Server in Node.js. You may not find as many tutorials and resources, but **your requirements should determine the database type** — *not the language*.

*(That said, don't make life purposely difficult for yourself! Choosing an unusual technology combination or a mix of SQL and NoSQL is possible, but you'll find it tougher to find support and employ experienced developers.)*

Remember - always use the right tool for the job