5.2 Code Execution
5.3 Monitoring

Tools

- Puppet
  - https://puppet.com/docs/puppet/6/puppet_overview.html
  - What is Puppet? | How Puppet Works? | Puppet Tutorial For Beginners | DevOps Tools | Simplilearn https://www.youtube.com/watch?v=llcjg1R0DdM
  - What is Puppet | Puppet Tutorial for Beginners | Puppet Configuration Management Tutorial | Edureka https://www.youtube.com/watch?v=PL_J5Gj3GAQ
- Ansible
  - https://www.geeksforgeeks.org/introduction-to-ansible-and-its-architecture-components/
  - https://docs.ansible.com/ansible/2.5/dev_guide/overview_architecture.html
  - What Is Ansible? | How Ansible Works? | Ansible Tutorial For Beginners | DevOps Tools | Simplilearn https://www.youtube.com/watch?v=wgQ3rHFTM4E

Tools

- Chef
  - https://www.tutorialspoint.com/chef/chef_tutorial.pdf
  - What is Chef in DevOps? | Chef Tutorial | DevOps Chef Training Video | DevOps Tools | Simplilearn https://www.youtube.com/watch?v=lqOJIenrwp0
- SaltStack
  - https://docs.vmware.com/en/VMware-vRealize-Automation-SaltStack-Config/8.6/use-manage-saltstack-config/GUID-FFD5DAE2-5FCD-46B2-8334-21799FB8FECE.html
  - Intro to SaltStack: DevOps Library SaltStack #1 https://www.youtube.com/watch?v=W0i5FpCq2WI

- **Chef vs Puppet vs Ansible vs Saltstack | Configuration Management Tools | DevOps Tools | Simplilearn**
- https://www.youtube.com/watch?v=_TVNCTK808I

Tools

- PalletOps
  - PalletOps is an advanced deployment system, which combines the declarative power of Lisp with a very lightweight server configuration.
  - PalletOps takes Ansible's agentless idea one step further. Rather than needing a Ruby or Python interpreter installed on the node that is to be configured, you only need ssh and a bash installation. These are pretty simple requirements.
  - PalletOps compiles its Lisp-defined DSL to Bash code that is executed on the slave node. These are such simple requirements that you can use it on very small and simple servers—even phones!
  - On the other hand, while there are a number of support modules for Pallet called **crates**, there are fewer of them than there are for Puppet or Ansible.
  - https://www.youtube.com/watch?v=2U0QdyWQAGM

Tools

- Vagrent
  - What is Vagrant? Introduction to Vagrant | Vagrant Tutorial for Beginners | DevOps Training
  - https://www.youtube.com/watch?v=Bv6-ClIitYs
- Docker
- AWS
- Azure

Comparison tables

Here is such a terminology comparison chart:

System	Puppet	Ansible	Pallet	Salt
Client	Agent	Node	Node	Minion
Server	Master	Server	Server	Master
Configuration	Catalog	Playbook	Crate	Salt State

Also, here is a technology comparison chart:

System	Puppet	Ansible	Pallet	Chef	Salt
Agentless	No	Yes	Yes	Yes	Both
Client dependencies	Ruby	Python, sshd, bash	sshd, bash	Ruby, sshd, bash	Python
Language	Ruby	Python	Clojure	Ruby	Python

Monitoring Tools

- Nagious
  - Nagios Monitoring Tool Tutorial | Server Monitoring with Nagios | DevOps Tools | Intellipaat
  - https://www.youtube.com/watch?v=NhAaEM-AsIw
- Munin
  - https://guide.munin-monitoring.org/en/latest/tutorial/getting-started.html
- Ganglia
  - https://www.digitalocean.com/community/tutorials/introduction-to-ganglia-on-ubuntu-14-04
  - Ganglia Performance Monitoring Across Machines https://www.youtube.com/watch?v=dF0uD-GVRl4
- Graphite
  - https://graphiteapp.org/
- Log Handling

Monitoring Tools

Munin

- To graph server statistics such as memory usage, which is useful in order to understand overall server health.
- Since Munin graphs statistics over time, you can see resource allocation trends, which can help you find problems before they get serious.
- It is designed to be easy to use and set up. The out-of-the-box experience gives you many graphs with little work.

Monitoring Tools

Munin

- While Nagios focuses on the high-level traits of the health of a service (whether the service or host is alive or not in binary terms), Munin keeps track of statistics that it periodically samples, and draws graphs of them.
- Munin can sample a lot of different types of statistics, from CPU and memory load to the number of active users in your children's Minecraft server. It is also easy to make plugins that get you the statistics you want from your own services.
- An image can say a lot and convey lots of information at a glance, so graphing the memory and processor load of your servers can give you an early warning if something is about to go wrong.

Monitoring Tools

Munin

- Like any of the monitoring systems we are exploring in this chapter, Munin also has a network-oriented architecture, as explained here. It is similar in design to Nagios. The main components of Munin are as follows:
- There is a central server, the Munin master, which is responsible for gathering data from the Munin nodes. The Munin data is stored in a database system called **RRD**, which is an acronym for **Round-robin Database**. The RRD also does the graphing of the gathered data.
- The Munin node is a component that is installed on the servers that will be monitored. The Munin master connects to all the Munin nodes and runs plugins which return data to the master.

Monitoring Tools

Ganglia

- Ganglia is a graphing and monitoring solution for large clusters. It can aggregate information in convenient overview displays.
- The word "ganglia" is the plural form of ganglion, which is a nerve cell cluster in anatomy. The analogy implies that Ganglia can be the sensory nerve cell network in your cluster.
- Like Munin, Ganglia also uses RRDs for database storage and graphing, so the graphs will look similar to the previous Munin graphs. Code reuse is a good thing!

Monitoring Tools

Ganglia

- Ganglia consists of the following components:
- **Gmond**: This is an acronym for **Ganglia monitoring daemon**. Gmond is a service that collects information about a node. Gmond will need to be installed on each server that you want Ganglia to monitor.
- **Gmetad**: This stands for **Ganglia meta daemon**. Gmetad is a daemon which runs on the master node, collecting the information that all the Gmond nodes gather. Gmetad daemons can also work together to spread the load across the network. If you have a large enough cluster, the topology does indeed start to look like a nerve cell network!
- **RRD**: A round-robin database, the same tool Munin uses on the master node to store data and visualizations for Ganglia in time series that are suitable for graphing.
- **A PHP-based web frontend:** This displays the data that the master node has collected and RRD has graphed for us.

Monitoring Tools

Ganglia

- Compared to Munin, Ganglia has an extra layer, the meta daemon. This extra layer allows Ganglia to scale by distributing the network load between nodes.
- Ganglia has a grid concept, where you group clusters with a similar purpose together. You can, for example, put all your database servers together in a grid. The database servers don't need to have any other relationship; they can serve different applications. The grid concept allows you to view the performance metrics of all your database servers together.
- You can try Ganglia out locally as well, again, using a Docker image.

Monitoring Tools

Graphite

- While Munin is nice because it is robust and fairly easy to start using, the graphs it provides are only updated once in a while, normally every fifth minute. There is therefore a niche for a tool that does graphing that is closer to real time. Graphite is such a tool.
- The Graphite stack consists of the following three major parts. It is similar to both Ganglia and Munin but uses its own component implementations.
- The Graphite Web component, which is a web application that renders a user interface consisting of graphs and dashboards organized within a tree-like browser widget

Monitoring Tools

Graphite

- The Carbon metric processing daemon, which gathers the metrics
- The Whisper time series database library
- As such, the Graphite stack is similar in utility to both Munin and Ganglia. Unlike Munin and Ganglia though, it uses its own time series library, Whisper, rather than an RRD.

Monitoring Tools

- **Log handling**
- Log handling is a very important concept, and we will explore some of the many options, such as the **ELK** (**Elasticsearch**, **Logstash** and **Kibana**) stack.
- Traditionally, logging just consisted of using simple print statements in code to trace events in the code. This is sometimes called **printf-style debugging**, because you use traces to see how your code behaves rather than using a regular debugger.

Monitoring Tools

- **Log handling**
- Here is a simple example in C syntax. The idea is that we want to know when we enter the function *fn(x)* and what value the argument *x* has:
- void fn(char *x){
- printf("DEBUG entering fn, x is %s\n", x);
- …
- }
- From the debug traces in the console, you can determine whether the program being developed is behaving as expected.

Monitoring Tools

- **Log handling**
- You would, of course, also like to see whether something serious is wrong with your program and report that with a higher priority:
- printf("ERROR x cant be an empty string\n");
- There are several problems with this style of debugging. They are useful when you want to know how the program behaves, but they are not so useful when you have finished developing and want to deploy your code.
- Today, there are a great number of frameworks that support logging in many different ways

Monitoring Tools

- **Log handling**
- The logging frameworks add value to printf-style logging primarily by defining standards and offering improved functionality such as these:
- Different log priorities, such as **Debug, Warning, Trace, and Error.**
- Filtering log messages of different priorities. You might not always be interested in debug traces, but you are probably always interested in error messages.
- Logging to different destinations, such as files, databases, or network daemons. This includes the ELK stack that we will visit later.
- The rotating and archiving of log files. Old log files can be archived.

Monitoring Tools

- **Log handling**
- Every now and then, a new logging framework pops up, so the logging problem domain seems far from exhausted even to this day. This tendency is understandable, since properly done logs can help you determine the exact cause of the failure of a network service that is no longer running or, of course, any complicated service that you are not constantly supervising. Logging is also hard to do right, since excessive logging can kill the performance of your service, and too little doesn't help you determine the cause of failures. Therefore, logging systems go to great lengths to strike a balance between the various traits of logging.

Monitoring Tools

- **Client-side logging libraries**
  - **Log4j** is a popular logging framework for Java. There are several ports for other languages, such as:
  - **Log4c** for the C language
  - **Log4js** for the JavaScript language
  - **Apache log4net**, a port to the Microsoft .NET framework
  - Several other ports exist, with many different logging frameworks that share many of their concepts.

Monitoring Tools

- **Client-side logging libraries**
- Since there are many logging frameworks for the Java platform alone, there are also some wrapper logging frameworks, such as **Apache Commons Logging** or **Simple Logging Facade for Java** (**SLF4J**). These are intended to allow you to use a single interface and swap out the underlying logging framework implementation if needed.
- **Logback** is meant to be the successor of log4j and is compatible with the ELK stack.
- Log4j is the first of the Java logging frameworks, and essentially gives you the equivalent of the previous printf statements but with many more bells and whistles.
- Log4j works with three primary constructs:
  - Loggers, Appenders, Layouts