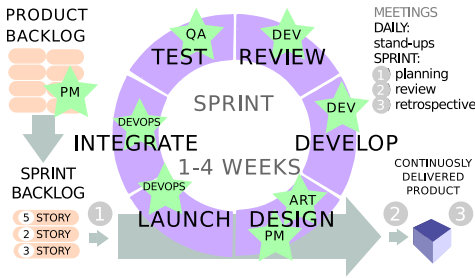


## SCRUM SPRINT DIAGRAM



## AGILE TERMINOLOGY

**Agile** Most popular modern methodology to organize software dev

**Story** All work is broken down into “stories” which constitute discrete features to be added or defects to be fixed (depending on org, might be *ticket*, *task*, *issue*)

**Story points** Rough estimates of work involved to perform a task, intentionally left unit-less

**Scrum** Popular organizational method that focuses on planning cyclic *sprints* of effort, and *velocity* (team effectiveness) is measured based on total *story points* performed each sprint

**Kanban board** Story completion showed in columns

**Kanban** Popular organizational method to produce a continuous flow of effort, and *cycle time* is measured based on average time it takes to complete stories

## ROLES

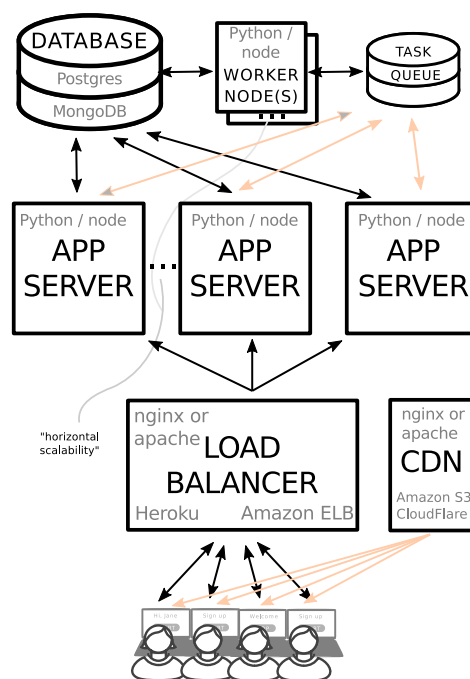
**Product Management** Role within an engineering team of converting business needs into user stories, doing product research, and producing concrete *stories* with specifications and wireframes

**Developer** Writes the code that implement a *story* about a new feature or bug fix

**QA Engineer** Inspect product for defects and regressions, and write tools to automate this process

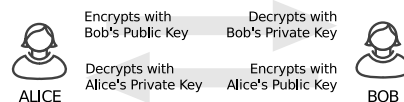
**DevOps** Sets up servers and DBs, and launches new code to production, automating the process to achieve “continuous deployment”

## SERVER TOPOLOGY



## ENCRYPTION

**Key pair** A pair of *private* and *public* key files to be used to securely connect a pair of computers online, used by git, ssh, and more



## DEVOPS TOOLS

**Virtual machine** Can simulate multiple computers on a single physical computer. Linux VMs can be bought at hosts such as Amazon Web Services or Digital Ocean.

**SSH** Remotely log into a Bash terminal of a Linux computer or VM

**Docker** Creates *containers* which isolate software and lock-down versions on a Linux system (e.g. anything apt installed on Ubuntu)

**Puppet, Chef, Ansible, Salt etc** Tools used by DevOps for running and configuring many servers

## AUTOMATED TESTS

**Unit testing** Tests a single function or class in isolation – colloquially sometimes refers to all automated tests

**Functional testing** Tests that a component performs the function that supposed to do

**End-to-end testing** Simulates clicking through a browser to ensure the app is fully working from a user standpoint

## JEST CHEATSHEET

```
x = // result of operation

// Strict equality
expect(x).toBe(42)
// Strict equality
expect(x).not.toBe(3)
// Deep equality
expect(x).toEqual([1, 2])
// Deep equality
expect(x).toEqual({ b: 2 })

// Matches anything true-like
// e.g. true, "test", 123
expect(x).toBeTruthy()
// Matches anything false-like
// e.g. false, 0, ""
expect(x).toBeFalsy()

// Numbers
expect(x).toBeGreaterThan(1)
expect(x).toBeGreaterThanOrEqual(1)
expect(x).toBeLessThan(2)
expect(x).toBeLessThanOrEqual(1)
expect(x).toBeCloseTo(0.3, 5)

// Strings (inclusion)
expect(x).toMatch("tea")
expect(x).not.toMatch("coffee")

// Arrays & Objects
expect(x).toHaveLength(3)
expect(x).toContain("Alice")
expect(a).toHaveProperty("a")
expect(a).toMatchObject({ a: 1 })

nightmare
.goto("yahoo.com")
.type("[name=p]", "dog memes")
.click(".SearchButton")
.wait("#main")
.evaluate(() => (
  document
    .querySelector("a").href
))
```