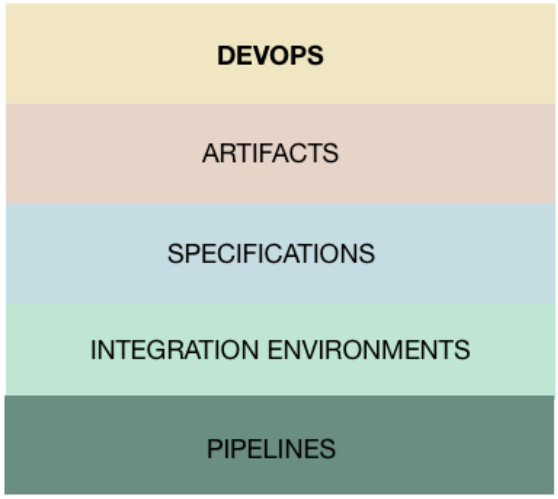


Ephemeral DevOps

As organizations become flatter and computing resources become more elastic, the combination of the these two trends provides new opportunities for cross functional teams. Ephemeral groups[1], or groups that arise and disappear based on needs other than command and control directives, combined with the lean behavior found in the DevOps community, provide democratized, just-in-time service development with little to no waste.

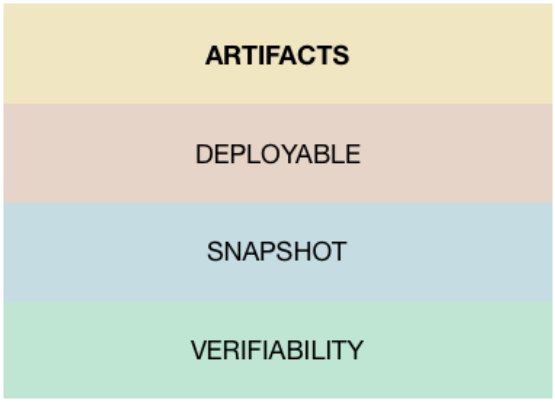
[1] Ephemeral Groups, W. Watson, 2016, https://s3-us-west-2.amazonaws.com/vulk-blog/EphemeralGroups.pdf

Copyright © 2016 W. Watson, Vulk LLC All Rights Reserved
http://www.vulk.coop



DevOps

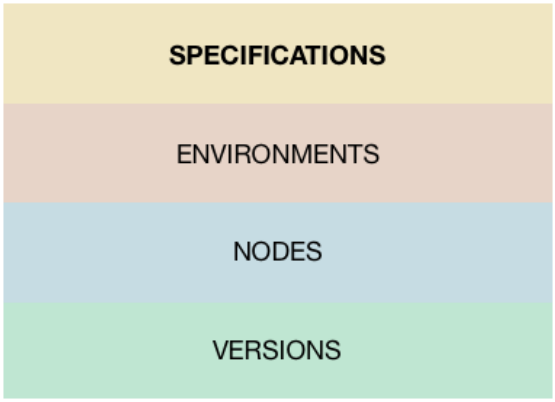
In order for us to understand DevOps at a high level, independent from the available tools, we need to understand the components that deliver value behind DevOps. At this very simplified level, these would be artifacts, specifications, environments, and pipelines.



Artifacts

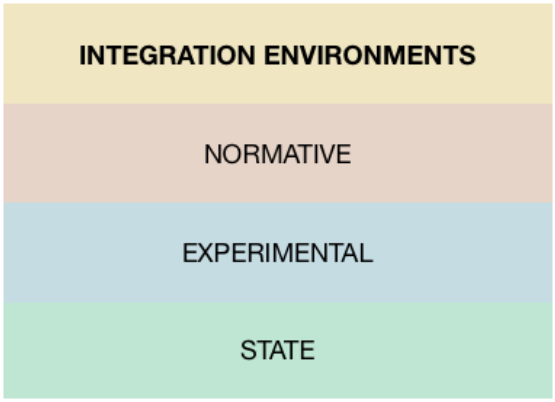
Artifacts are deployable, verifiable, and represent a snapshot in time (the time of deployment). This way of viewing architecture, the view that only the structure of actually deployed executables (not the structure of code itself) is architecture, helps us to reason about the tradeoffs of non-functional requirements. This is because artifacts identify the coarse grained, reusable, and composable parts of the system. Artifacts include binaries, images, scripts (executable code, data, and schemes), and hashes of any or all of these resources. The hashes are used to verify that actually deployed artifacts are the same as stored artifacts in the event that the stored artifacts have to be deleted to save space.

Copyright © 2016 W. Watson, Vulk LLC All Rights Reserved
http://www.vulk.coop



Architecture specifications identify how nodes, specific versions of code, and specific ways of deploying that code are combined to create an environment. Specifications refer to environment configurations, toolchain configurations, and all other application configuration. Within an architecture specification, the environment's application configuration is the structure of its nodes (i.e. servers) and usually has a title. Some examples of titles are 'Development','Test', or 'Production'. Preferably an application configuration should be same for all environments but there can be some differences (e.g. The 'staging' environment gets a 'production' data load and has additional tests). The specification must know which artifacts reside on which nodes and they must know the networking configuration of each node. The specification also knows the specific version, preferably through semantic versioning, of the code that deploys the artifacts. This is not to be confused with the specific version of the deployed resources themselves (e.g. Apache version 2.4).

There can be a community version of a deployment script which can have its own version and in some cases is customized by the deployment team. In other cases the community version of a deployment script is so volatile when combined with other artifacts in the environment that trusting automatic minor version upgrades is too risky. In order to account for this there must be some way of 'pinning' the specific version of a deployment script, whether community originated or customized, to a specific deployment. This allows for maximum upgrading during routine deployments and tests, which allows for more frequent security upgrades, but also reduces risk of breaking the build.



Environments are composed of artifacts and specifications, and usually have titles that are familiar to most technical people (e.g. development, test, production). The normative names are development (other developers can build and use the environment), test (QA professionals can use the environment), staging

Copyright © 2016 W. Watson, Vulk LLC All Rights Reserved
http://www.vulk.coop

(test code is merged with production data), and production (customers can use the environment). In addition to normative environments there can be experimental builds which are used for things like code spikes, version upgrade tests, or performance architecture tests. There are two states that an environment can be in. When an environment is in a planned state, it is in the process of being built and is not available to its regular consumers. When an environment is completed, all work towards the build (including automated tests, QA approval, and code reviews), is completed and it is available to its consumers. For instance, when the 'test' environment is being built, there is a current 'test' environment that will be replaced if and when the new planned 'test' environment is complete. Until then, only the original test environment is available to its consumers. The conversion from the planned environment to completed environment is called a blue/green deployment and addresses environmental availability in the build process.



Pipelines

Pipelines are a description of the signals that push artifacts along pathways (aka stages) between environments. The time and effort that it takes to push things through a pipeline is a measure of its efficiency. Signals (aka triggers) cause environments to be built. The signal that says that an environment has changed creates a new environment and puts it into a 'planned' or 'in-process' state. Some examples of signals are code check-ins, new artifact modifications, software upgrades, a test suite passing, and a manual test passing. The pathway between environments in a pipeline are described using sequence tags (dev -> test -> staging -> production), where the artifacts tagged in a repository are moved along the prescribed direction in the pipeline. When a build signal is received, a new planned environment is created. When a merge signal is received a planned environment turns into a completed environment and replaces its corresponding tagged environment.

Pathways

A common pathway, or set of stages within a pipeline are as follows[2]:

1. Commit stage
2. Automated acceptance stage
3. Manual Test stage
4. Release stage

[2] Humble, Jez, and David Farley. *Continuous Delivery*. Addison-Wesley, 2011.

Copyright © 2016 W. Watson, Vulk LLC All Rights Reserved
http://www.vulk.coop

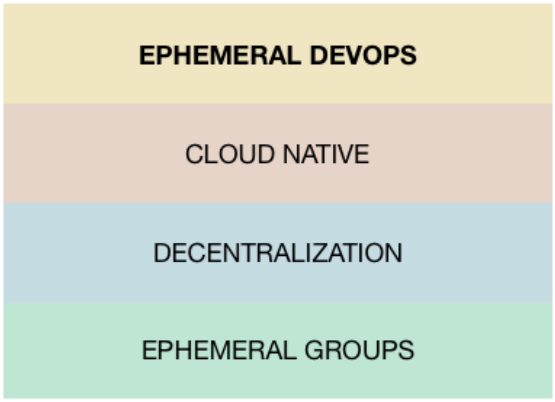
At each stage, the appropriate tests are run and a public pass/fail indicator is made available through a pipeline dashboard. This allows for coordination within and between teams based on the readiness of a specific commit.

Efficiencies and Risk

Automated pipelines promote frequent deploys which helps an organization to practice moving code into new environments while applying the relevant tests. Given that the risk of a deployment is the inverse of how many times you deploy, pipelines with click button deploys reduce risk and increase efficiency.

Pull System

When a deployment pipeline is implemented, the software development behavior changes into a pull system. A pull system is one in which testing, operations, or development autonomously deploy into the environments that they are responsible for. This self-service development style minimizes the amount of time that anyone is waiting for a working build to be deployed. Along with this increased access, increased visibility is given to everyone interested in the status of a commit as it works its way through the build process.



Ephemeral DevOps is an approach for delivering features that develop cloud native applications using a decentralized cross-functional team which may only exist for the time it takes to deliver a specific customer need.

Cloud Native

A cloud native application lends itself to ephemeral groups because of its commitment to elasticity, hasty yet reliable environments, and promotion of microservice development. An ephemeral group that emerges within an organization must deliver a reliable service, often with a service level agreement, quickly. The cloud native community is focused on building environments and tools that require reproducibility, resource isolation, transparency, monitored failures, and distributed data caches which are necessary for reliable service development. Tools like Kubernetes and Docker allow for node orchestration in a standard way, allowing an ephemeral group to concentrate on delivery of a reliable service.

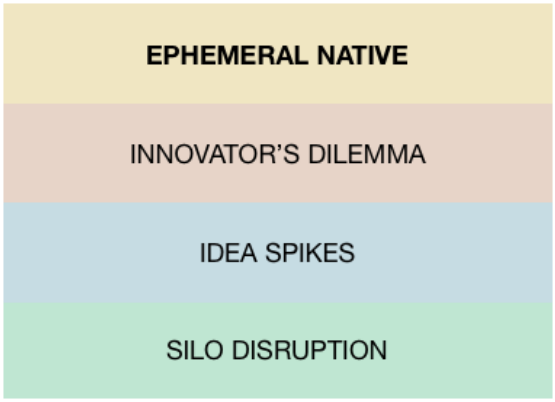
Copyright © 2016 W. Watson, Vulk LLC All Rights Reserved
http://www.vulk.coop

Decentralization

A decentralized team is one with little hierarchy. This happens when an experienced group, often with complementary skill sets, emerges to solve a problem or fulfill some need. Traditional command and control would assemble a group from the top down, with traditional incentives of pay, promotion, etc. A decentralized team autonomously emerges based on more natural incentives, such as a customer need. This may include direct compensation from the customer, in order to improve or increase their reputation, or simply because the service or feature needed to be offered in order to solve another far reaching goal. When the incentive for action is directly tied to customers or immediate needs, the knowledge of requirements is pushed down into the decentralized group. This makes the group more responsive to changes in requirements for the service being provided.

Ephemeral Groups

An ephemeral group is a coalition that comes into existence for just enough time to pursue an opportunity and then dissolves. Ephemeral groups can be formed within a larger organization or from a set of individuals with little previous knowledge of one another[3].



Ephemeral native

Given the elastic nature of cloud native environments and ephemeral groups, we can coin a new phrase, ephemeral native, that denotes applications and groups that are spun up to respond directly to an immediate need and then spun back down when the service is delivered.

Innovator's dilemma

One of the reasons why initiatives need to emerge autonomously, in a decentralized way is because of the innovator's dilemma. This problem is defined as:

1. Good management obeys customer desires[4]

---

[3] Ephemeral Groups, W. Watson, 2016, https://s3-us-west-2.amazonaws.com/vulk-blog/EphemeralGroups.pdf

[4] "What often causes this lagging behind are two principles of good management taught in business schools: that you should always listen to and respond to the needs of your best customers, and that you should focus investments on those innovations that promise the highest returns. But these two principles, in practice, actually sow the seeds of every successful company's ultimate demise.",

Copyright © 2016 W. Watson, Vulk LLC All Rights Reserved
http://www.vulk.coop

2. Customers desire more of the same
3. Therefore, good management obeys the desire for more of the same.

The paradox for innovation lies in that *innovation is not more of the same*. Ephemeral groups, when paired with the lighter resource requirements provided by a cloud native environment, have less resistance when developing services that a customer *wants* but *did not ask for* specifically.

Silo disruption

With ephemeral development, another opportunity emerges with respect to silos. Silos emerge when a group, for political reasons or unintentionally, does not share information or resources with other groups. When the resources for development are so small so as to not require a major sign off, enterprising people can cross the divide with working solutions before the red tape can stop them.

Conclusion

In the past, for smaller projects, enterprising individuals would have the resources to go against the grain and take initiative. Now that DevOps has made reliable production level environments available at the push of a button, the threshold has been lowered so that larger cross functional teams that concentrate on even bigger problems can innovate more easily as well. The creation of deployment pipelines have made the build, test, and deploy status of code publically available so that multiple cross functional teams can coordinate service development down to a specific commit. These factors, along with more autonomy in the workplace, allow for groups to come together in a just-in-time fashion to coalesce and use just enough resources to meet the changing demands of customers.

Christensen, Clayton. The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail (Management of Innovation and Change) (Kindle Locations 91-94). Harvard Business Review Press. Kindle Edition.

Special thanks to the Vulk team

Copyright © 2016 W. Watson, Vulk LLC All Rights Reserved
http://www.vulk.coop