Key Concepts in PCA

1. **Dimensionality Reduction**:
    - PCA is often used to reduce the number of variables in a dataset (dimensionality) while retaining as much information as possible.
    - This is particularly useful when dealing with high-dimensional data, where many variables are correlated and redundant.
2. **Eigenvectors and Eigenvalues**:

- PCA involves finding the eigenvectors and eigenvalues of the covariance matrix of the data.
- The eigenvectors (principal components) indicate the direction of maximum variance, while the eigenvalues indicate the magnitude of variance in that direction.

Docker cycle and tools

Discover

In the Discover phase, a DevOps team researches and defines the scope of a project. In particular, it involves activities such as user research, establishing goals, and defining success.

Plan

Taking a page out of the agile handbook, we recommend tools that allow development and operations teams to break work down into smaller, manageable chunks for quicker deployments. This allows you to learn from users sooner and helps with optimizing a product based on the feedback. Look for tools that provide sprint planning, issue tracking, and allow collaboration, such as Jira.

Build

Production-identical environments for development

While Puppet and Chef primarily benefit operations, developers use open source tools like Kubernetes and Docker to provision individual development environments. Coding against virtual, disposable replicas of production helps you get more work done.

Infrastructure as code

Developers create modular applications because they're more reliable and maintainable. So why not extend that thinking to IT infrastructure? This can be difficult to apply to systems because they are always changing. So we get around that by using code for provisioning.

Infrastructure as code means re-provisioning is faster than repairing – and more consistent and reproducible. It also means you can easily spin up variations of your development environment with similar configuration as production. Provisioning code can be applied and reapplied to put a server into a known baseline. It can be stored in version control. It can be tested, incorporated into CI (continuous integration), and peer-reviewed.
When institutional knowledge is, well, codified in code, the need for run books and internal documentation fades. What emerges are repeatable processes, and reliable systems.

Source control and collaborative coding

It's important to have source control of your code. Source control tools help store the code in different chains so you can see every change and collaborate more easily by sharing those changes. Rather than waiting on change approval boards before

deploying to production, you can improve code quality and throughput with peer reviews done via pull requests.

What are pull requests, you ask? Pull requests tell your team about changes you've pushed to a development branch in your repository. Your team can then review the proposed changes and discuss modifications before integrating them into the main code line. Pull requests increase the quality of the software which results in less bugs / incidents, which reduces operational costs and results in faster development.
Source control tools should integrate with other tools, which allows you to connect the different parts of code development and delivery. This allows you to know if the feature's code is running in production. If an incident occurs, the code can be retrieved to shed light on the incident.

Continuous Delivery

Continuous integration is the practice of checking in code to a shared repository several times a day, and testing it each time. That way, you automatically detect problems early, fix them when they're easiest to fix, and roll out new features to your users as early as possible.

Test

Automated testing

Testing tools span many needs and capabilities, including exploratory testing, test management, and orchestration. However, for the DevOps toolchain, automation is an essential function. Automated testing pays off over time by speeding up your development and testing cycles in the long run. And in a DevOps environment, it's important for another reason: awareness.
Test automation can increase software quality and reduce risk by doing it early and often. Development teams can execute automated tests repeatedly, covering several areas such as UI testing, security scanning, or load testing. They also yield reports and trend graphs that help identify risky areas.

Risk is a fact of life in software development, but you can't mitigate what you can't anticipate. Do your operations team a favor and let them peek under the hood with you. Look for tools that support wallboards, and let everyone involved in the project comment on specific build or deployment results. Extra points for tools that make it easy to get Operations involved in blitz testing and exploratory testing.

Deploy

Deployment dashboards

One of the most stressful parts of shipping software is getting all the change, test, and deployment information for an upcoming release into one place. The last thing anyone needs before a release is a long meeting to report on status. This is where release dashboards come in.

Look for tools with a single dashboard integrated with your code repository and deployment tools. Find something that gives you full visibility on branches, builds, pull requests, and deployment warnings in one place.

Automated deployment

There's no magic recipe for automated deployment that will work for every application and IT environment. But converting operations' runbook into a cmd-executable script using Ruby or bash is a common way to start. Good engineering practices are vital. Use variables to factor out host names – maintaining unique scripts or code for each environment is no fun (and misses half the point anyway). Create utility methods or scripts to avoid duplicated code. And peer review your scripts to sanity-check them.

Try automating deployments to your lowest-level environment first, where you'll be using that automation most frequently, then replicate that all the way up to production. If nothing else, this exercise highlights the differences between your environments and generates a list of tasks for standardizing them. As a bonus, standardizing deploys through automation reduces "server drift" within and between environments.

Operate

Incident, change and problem tracking

The keys to unlocking collaboration between DevOps teams is making sure they're viewing the same work. What happens when incidents are reported? Are they linked and traceable to software problems? When changes are made, are they linked to releases?

Nothing blocks Dev's collaboration with Ops more than having incidents and software development projects tracked in different systems. Look for tools that keep incidents, changes, problems, and software projects on one platform so you can identify and fix problems faster.

Observe

Application and server performance monitoring

There are two types of monitoring that should be automated: server monitoring and application performance monitoring.

Manually "topping" a box or hitting your API with a test is fine for spot-checking. But to understand trends and the overall health of your application (and environments), you need software that is listening and recording data 24/7. Ongoing observability is a key capability for successful DevOps teams.

Look for tools that integrate with your group chat client so alerts go straight to your team's room, or a dedicated room for incidents.

Continuous Feedback

Customers are already telling you whether you've built the right thing – you just have to listen. Continuous feedback includes both the culture and processes to collect feedback regularly, and tools to drive insights from the feedback. Continuous feedback practices include collecting and reviewing NPS data, churn surveys, bug reports, support tickets, and even tweets. In a DevOps culture, everyone on the product team has access to user

comments because they help guide everything from release planning to exploratory testing sessions.
Look for applications that integrate your chat tool with your favorite survey platform for NPS-style feedback. Twitter and/or Facebook can also be integrated with chat for real-time feedback. For deeper looks at the feedback coming in from social media, it's worth investing in a social media management platform that can pull reports using historical data.

Analyzing and incorporating feedback may feel like it slows the pace of development in the short term, but it's more efficient in the long run than releasing new features that nobody wants.