TEN STEPS

to DevOps at Scale

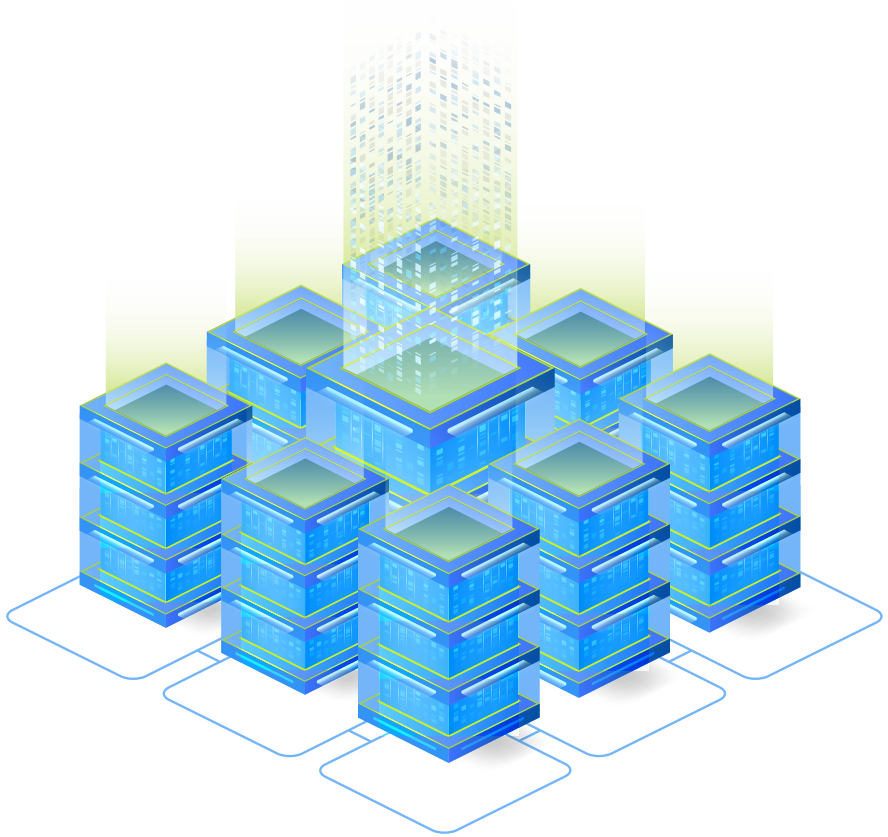
NUTANIX™
YOUR ENTERPRISE CLOUD

FROM PETS TO HERDS, BESPOKE TO EPHEMERAL

DevOps is a powerful methodology that enables businesses to deliver better quality products and services to customers, both faster and at a lower cost. It achieves these benefits by radically departing from traditional IT service delivery processes. To make the DevOps transformation more easily understood, this eBook presents 10 steps describing the move from a highly traditional, siloed, and "monolithic" IT organization to continuous enterprise cloud deployment and true hybrid cloud facility.

Decades of habits and traditions can make it difficult to see alternative strategies for solving problems, especially when there's significant risk involved. Until fairly recently, it's been taken as a given that medium to large companies organized themselves into separate silos of functional expertise and responsibilities, with projects following a linear "waterfall" process. But legacy approaches have proven not only slow, inefficient, and costly, they also lead to buggy, substandard products that disappoint customers (or worse) and wreak havoc on the production environment. These shortcomings have become even more glaring with the advent of virtualization, distributed systems, and cloud computing. The search for agility has prompted some to go "all-in" on public cloud, but for most workloads, this is an extremely expensive way of doing business (as borne out by monthly sticker shock) and can end up being another form of vendor lock-in.

A reasonable person might ask, why do companies need DevOps when IT and Operations are successfully delivering security and uptime, using administrative tools to maintain, backup, and restore those systems? The answer is that if Operations is ever going to become a force multiplier that helps the business drive innovation, instead of a cost center whose primary function is to keep the lights on, it must become agile. For this to happen, Operations must be documented, democratized, and distributed.



The traditional values and practices for creating stable, long-lasting infrastructure do not apply when progressing to a DevOps mindset. IT infrastructure resources, both technological and human, have long been treated as unique and special--that is, treated like irreplaceable and extremely valuable “pets.” For example, most IT shops rely on servers that are bespoke (custom built) and maintained laboriously by hand. But these resources also constitute single points of failure that, when unavailable, can bring a mission-critical application, or even an entire organization, to its knees.

Conversely, DevOps requires that your material resources be distributed, fungible, and ephemeral. When one resource goes down, another takes its place without any disruption to the business. Technical resources like servers are more like members of a “herd” or fleet--one is as good as another. In terms of staffing, DevOps also democratizes required skill sets, so that the company is not dangerously reliant on any single “heroic” resource. The overall system should be smart enough so that no single employee possesses such specialized knowledge that their absence would significantly interrupt the flow of work. DevOps also dictates that, whenever possible, businesses should automate menial, error-prone, and repeatable tasks, freeing staff to devote their time and energy towards work that brings real value to the business.

Zero to DevOps:
The 10 Steps

For our model environment, we will explore, deconstruct, and rebuild a typical web application to cloud-like web scalability.

Our scenario starts with a bare-metal web application hosted on a Windows server, with Internet Information Services (IIS) (an extensible web server) and MS-SQL database. To minimize the variables, our example focuses exclusively on application infrastructure (a SAN or backup system target is out of scope).

Throughout the discussion, we posit a mythical, relative value of X as a unit of work and time, designated as 3 hours.

01

THE MONOLITH IN PRODUCTION

The journey begins with a hand-built prototype consisting of a web and database on a single server in a single datacenter. All updates happen in production. Restoring from backups are the only means to remediate the entire state of the application, configuration, data, and server. While there are many simple operational workarounds and procedures that can mitigate small problems, the application and infrastructure remain a monolith. There is a single point of failure for everything (architecture, operations, application, and updates), which precludes collaboration without the risk making easy but consequential mistakes. All single points of failure must be removed to progress from this stage toward scalability.

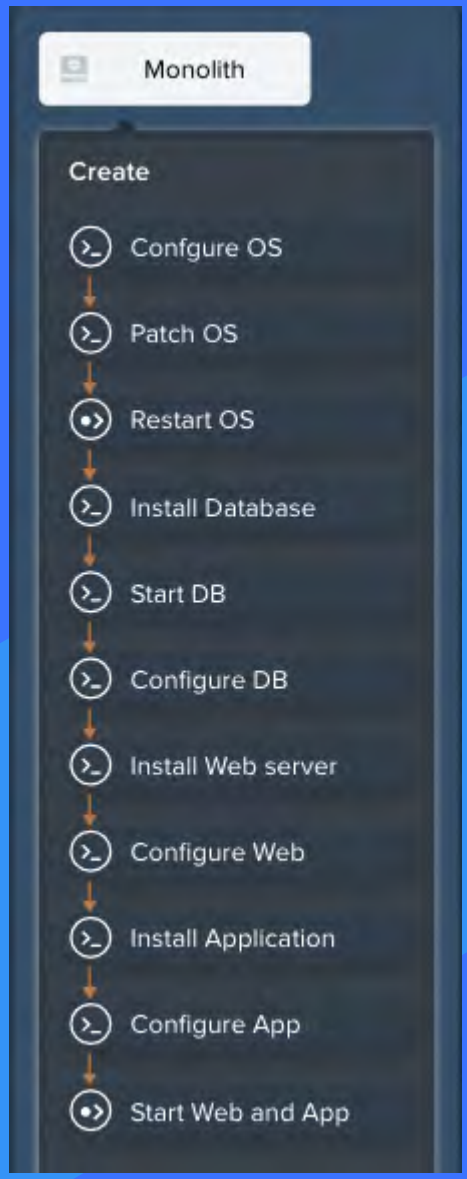
Design:
- A single server, hosting the entire application: OS + web + DB

Build and Deploy: 1X
- Built by hand

MTTR (Mean Time To Repair): 0.5X
- Repair is a restoral, estimated at 0.5 build and deploy time

Benefits:
- Easy to backup, but requires application downtime

Disadvantages:
- A single point of failure on every infrastructure and application tier
- A single point of failure environment: no staging
- A single point of failure datacenter: no disaster recovery
- Downtime requires human intervention to restore



02

SEPARATE, MONOLITHIC FUNCTIONS

The first step is to take the prototypical application and infrastructure apart. By separating the monolith into two parts, where each part has its own functional domain (web or database), the amount of work doubles. However, this work lays the foundation for isolating failure and distributing risk--one move away from the situation where any fault affects the entire system.

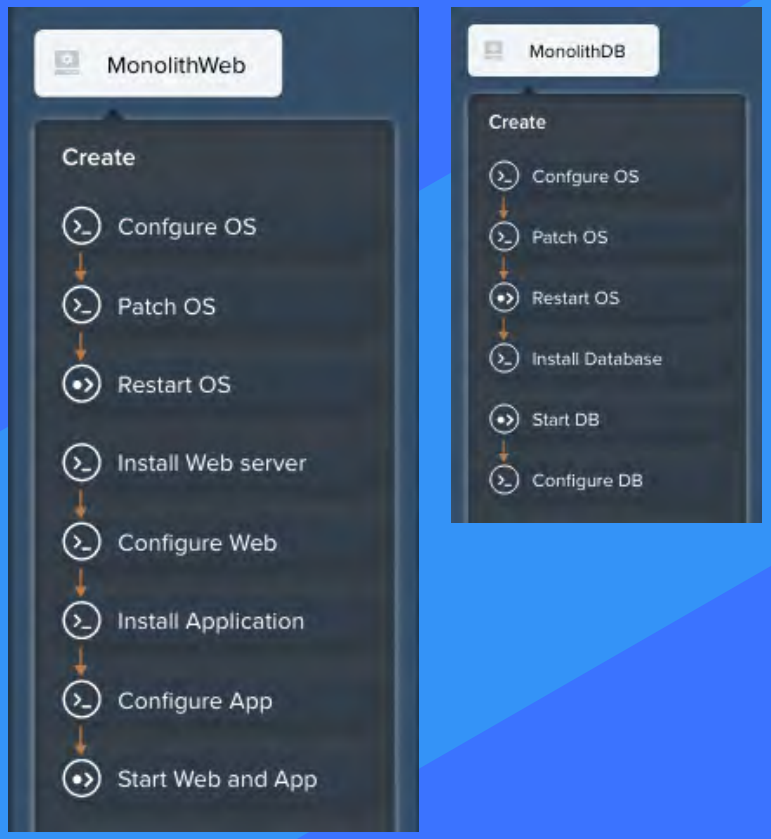
Design:
- Two servers in production:
  - Web hosts the application: OS + web
  - Database hosts the data: OS + DB

Build and Deploy: 1X
- Built by hand, thus little parallel OS installation time savings

MTTR: 0.5X
- Repair is a restoral, estimated at half the build and deploy time of this step

Benefits:
- Separation of each application function allows better management of each concern. There are two servers--if one goes down, the remediation scope is smaller than both database and web combined. The whole app is effectively down, but not all of the app is impacted.
- Easier to backup web without downtime
- Remediate Stage 1 disadvantage: if web node is down, DB is not, and vice-versa
- Minor MTTR decrease for restoral

Disadvantages:
- Each host remains a single point of failure
- Twice the time to maintain, twice the attack surface, twice the OS license cost
- Database maintenance still requires application downtime



03

LOAD BALANCING FOR UPTIME

The next step doubles the web and database server count into their respective tiers, and introduces a web load balancer. This breaks the direct sequence of dependencies between the web and database tiers. It allows updates on one instance, while the other handles double the demand during the maintenance window and preserves uptime for the application.

A further security refinement would be to place a demilitarized zone (DMZ) firewall between the web and database tiers, cutting off direct access and attacks to the database.

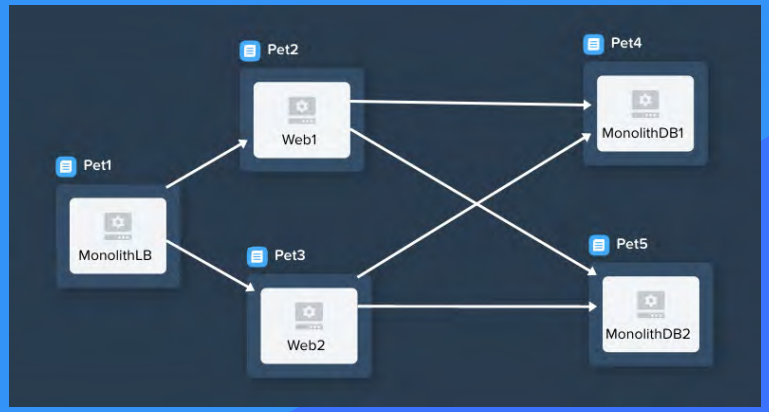
Design:
- Five servers:
  - Load balancer for the web tier, modeled with IIS Web Farm as a reverse proxy
  - Web tier: 2* web hosts the application: OS + web
  - Database: 2* database hosts the data: OS + DB

Build and Deploy: 6X
- 4X: Built by hand, so little parallel OS installation time savings
- 1X: Increased time to configure Master+Minion database
- 1X: Increased time to configure OS firewalls, web proxy

MTTR: 3X
- Repair is a restoral, estimated at half the build and deploy time of this step

Benefits:
- A single server in the web tier can fail or be maintained without application downtime
- In theory, a master and minion database can be maintained without downtime
- Potential 2x increased capacity for web and DB
- Application read traffic can be directed to minion to reduce load on master
- Reduced network attack surface:
- Web-tier OS firewall can restrict public access to load balancer
- Database OS firewall can restrict access to web-tier servers

Disadvantages:
- Load balancer is a single point of failure
- 5x the time to maintain; 5x OS license cost; 2x DB license costs
- MTTR is larger
- Increased complexity to troubleshoot network
- No DMZ network for further isolation



DECONSTRUCTING THE WORKLOAD "LIFT AND SHIFT" TRAP

The most common strategy for migrating workloads among data-centers, infrastructure providers, hypervisors, clouds (or even from physical to virtual machine, or from virtual machine to container) is through a storage conversion. This strategy is often called "lift and shift," because it represents the equivalent of a fork-load uplift of the entire captured state of the machine--moving the disk image (and any potential conversion) and dropping it off at the new destination to compalete reinstantiation.

"Lift and shift" is the easiest strategy to adopt because it can be accomplished with backup and restoral procedures and tooling. Most IT practitioners are familiar with this process given that,

1. There is an entire industry dedicated to storage backups.
2. Backup and restoral should be a standard operation for most IT organizations.
3. Backups are often a component for disaster recovery, which is another critical facility for IT organizations.

However, when used as a migration strategy, storage backup and restoral is limited to destination environments that are practically identical to the source environment. In essence, the only difference is location, because you want to preserve all other infrastructure characteristics to guarantee that workloads can run without further alteration in the destination environment. Even with minimal difference at the destination, there are many constraints that prevent a workload from running, including hard-coded dependencies on external service requirements (for example, DNS naming, IP addresses, network routes, firewalls, and VLANs, software and operating system licensing enablement). Furthermore, applications with hard-coded configurations may also pose a barrier if there is no software maintenance, lost source code and build environments, and programming skill sets for legacy applications. These dependencies and constraints represent additional operational overhead, unrecognized technical debt, and a pet mentality that blocks migration to other infrastructures.

Finally, the additional organizational overhead of maintaining an inventory of storage backups and their lifecycle (from near-line secondary storage to distant and off-line archival retrieval) is costly for these large backup files, even when outsourced.

Migrating workloads among different providers is a common yet onerous new requirement, especially when an organization has a "cloud-first" initiative.The second stage of the lift and shift strategy includes several time-consuming steps:

- Fulfill secure network transfer requirements between destinations
- Convert each workload's backup to the destination provider virtual machine format
- Transfer and provision the workload in the new provider.

Combining multiple workloads of large backup files over the network with conversion and downtime considerations, in addition to all of the previously mentioned operational overhead to remedy pet dependencies, may be an unacceptable amount of risk, cost, and disruption to the business. As a result of these hardships, many organizations are reconsidering "cloud first," moving instead to "cloud smart" to rationalize workload migration.

The ultimate issue with a lift and shift strategy is that it is a large, complex state transfer that entails the entire operational effort and history to maintain the workload up to the date of the snapshot or backup. The backup preserves every undocumented

change control and pet operation done by hand, such as adjustments to the operating system, application, and server configuration, security updates, and patches. Of course, any undetected nefarious, bad practices, or temporary and accidental changes, are also preserved and these changes are compounded over the lifecycle of years for a typical workload. Without extreme diligence in tooling and audits for change control operations, the state of the workload is unknowable. A large auditing industry exists around business documentation and enforcement of operational procedures for change, but short of a disaster recovery event, no audit can empirically test and validate those procedures.

Most IT organizations face an uncomfortable truth: disaster recovery is hard to achieve and expensive to operate. Most businesses end up relying on snap-shot cloning or backup restoral, which always incur the debt and history of their operational legacy.

Reproducing every historical operation and change from scratch is extremely burdensome, potentially incurring even more downtime than the second stage "lift and shift" strategy. Most IT organizations rely on a single approach for business continuity, one that involves large, complex state backups that contain unreproducible operations. Worst of all, this approach locks the business into the incumbent infrastructure providers who have no incentive to remove this constraint. Consequently, most IT organizations are trapped by their traditional tooling, vendors, and methods to solve the problem of how to run their business in a robust fashion.

The "cloud first" mantra of lifting and shifting workloads to the public cloud has often led to operational expenditure overspend. The unfortunate outcome can be an embarrassing second lift and shift back to on-premise facilities after a re-evaluation of cloud strategy. According to **NIST Special Publication 800-145**, cloud is multimodal in terms of characteristics, consumption, and operation. This perspective underwrites the **United States Federal Cloud Strategy**, which calls for a "cloud smart" rather than "cloud first" strategy. Simply put, application rationalization requires that you determine the proper cloud model for operation, which in turn calls for a mature hybrid-cloud with blended capital and operational expenditure outcomes.

Escaping the lift and shift trap requires paying down technical debt and adopting a new approach that confers the following strategic advantages:

1. **Reproducibility**: synthesizes workloads from scratch, shedding legacy and historical operations

2. **Portability**: instantiates the same workload on multiple providers, thereby achieving cloud smart initiatives

3. **Consistency**: facilitates testing and rollout of best practices, security patches, updates, and new facilities

4. **Automation**: combines reproducible, consistent operations with workload portability. Automation constantly improves the state of the business while also accelerating time to market, improving resource management efficiency.

IT is now evolving toward codifying infrastructure and operations and applying software engineering disciplines, which should make resource governance and audits implicit. DevOps encompasses the methods and cultural values required to achieve this transformation.

APPLICATION AUTOMATION AND ORCHESTRATION

This is where the transition from physical to virtual (P2V) begins. With the right tools, you can now automate and orchestrate the provisioning and deployment of your applications. State of the art automation tools can model business across application topology, orchestration dependencies, infrastructure configuration, and lifecycle operations. A common industry term for this business automation model is a "blueprint."

Document your current production deployment by modeling "brownfield," existing servers in a blueprint. Be sure that the blueprint documents lifecycle-orchestrated operations for every change control or runbook procedure (including, for example, backups, restorals, or an OS upgrade). Adopt role-based access control (RBAC) definitions to enable delegation of application deployment, including audits to delegate operations. Maintain blueprints with software revision control to ensure no loss to documented operations and infrastructure.

Stage 4 moves you away from hand-built provisioning of infrastructure and toward repeatable, no-error server deployment. It puts you on the road to infrastructure as code and marks the transition to becoming a developer: **programming ahead!**

Stage four is where an organization's DevOps journey really begins, as it breaks away from the "lift and shift" trap.

Design: a translation of stage 3
- 5 servers, brownfield imported

Deploy: 1X
- Using the Deploy estimate from the previous stage (6X), estimating automation to cut down on manual operations and human error at 33% savings, then parallelization at half of that.
- (6X/3)/2 = 1X, meaning we get 6 times the amount of work done as the original build and deploy work unit amount in stage 1.

Development: 4X
- 2X: Learning and bootstrapping an orchestration automation tool.
- 1X: Initial blueprint development time.
- 1X: Minimal because brownfield import, then adding actions for each documented change control or runbook operations. Should be cut and paste, with relatively little variable or macro substitution and minimal orchestration. Another huge step toward efficiency and automation.

MTTR: 0.5X
- Repair is a restoral, estimated at half the deploy time of this step.

Benefits:
- Documented production
- Added maintenance operations, can be delegated and audited under RBAC

Disadvantage:
- Still relatively rudimentary

THE TURNING POINT: WHAT'S AHEAD?
- Aspire towards creating push button, separate environment deployments. At this stage, however, still relying on the legacy approach of restoring in parallel and localizing through clones and snapshots. This will be stage 5.
- Add a revision control service. Paves the way to continuous integration of blueprints, infrastructure, and applications. This will be stage 8.
- Adding monitoring, metrics, and logs allows you to heuristically trigger these operations: this improvement moves towards zero-click operations, and the ideal of a self-healing, anti-fragile business. This will be stage 10.

05

SYNTHESIS WITH SCRIPTING: PUSH BUTTON ENVIRONMENTS

The goal of this stage is to change your brownfield, existing workloads (which have a lot of hard-coded configuration, data, and state in them) to push-button simplicity for new workload instances. How often does your software development and test team ask for a new environment and then wait around for days and weeks for IT to manually fulfill the requests? Nobody can afford to have developers wait to become productive anymore. With the advent of continuous integration, the demand for a new environment could happen with every developer check-in, multiple times per hour.

Cloning a VM template and powering on a new copy requires reconfiguration. Doing this manually for multiple VMs (for a database, a web server, a load balancer, etc.) is labor intensive and time consuming due to the existing state of each VM, which requires considerable reconfiguration and clean up.

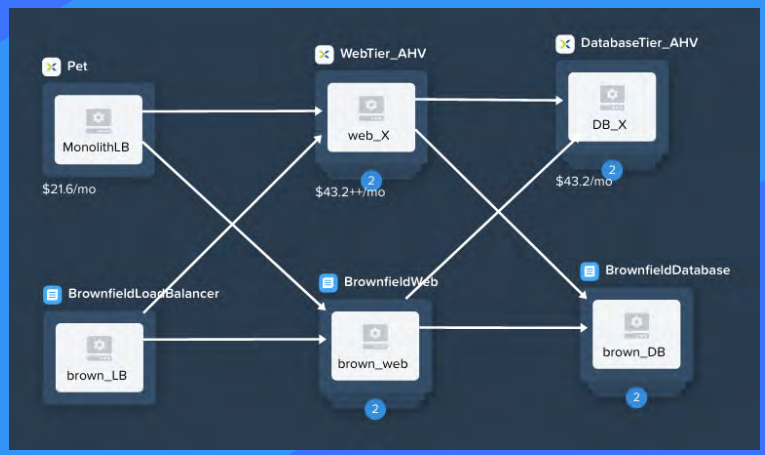
Using the brownfield blueprint as a starting point, you can create parallel instances of each VM and have it provisioned from scratch with a fresh template or operating system image, removing any pre-existing state and ensuring a pristine environment that is fully reproducible. You can also make the reconfiguration steps you performed by hand on the brownfield VMs into procedures to compose the newly provisioned VMs. There may be a need for external sources of configuration and data; we address this topic in the next stage with configuration management.

Once you have the ability to create the application VMs in parallel to the brownfield VMs, you can delete the brownfield VMs from the blueprint. What remains is a push-button application environment that you can delegate to end-users--not unlike the Apple App Store or Google Play Store. It is easy to create a new environment for your workloads to test new procedures, code, configuration, and operational changes, as well as OS and security updates and patches before working in production.

Another important refinement at this stage is to change hard-coded configuration information into variables that are determined at runtime, sourced eitherfrom external systems or your end-users.

Perhaps the most important achievement would be the transition from modeling each individual member (pet) and reorganizing them into appropriate tiers (herd) that can scale-in and scale-out. In other words, in lieu of two individual, pet web servers, you have a web-tier of servers that is initially deployed as a fleet (herd) population of two. This change, combined with variable configuration, provides the pivotal gateway to scalability.

This shift also requires orchestration, which coordinates the order of operations across the entire application architecture for dependencies. For example, the database must be created, configured, loaded with data, and ready before the web tier can connect to it.



The development and testing of this stage is 10x the original work unit, because it represents paying off technical debt to achieve a basic level of automation. For many, it is culturally unacceptable to devote this much time unless there is a plan and justification of effort. Crossing this hurdle fundamentally separates the maturity of every organization's operating model.

Design:
- A copy of stage 4, adding a parallel, synthesized workload with variable-driven configuration and application tiers, followed by brownfield removal.
- 5 servers, synthesized from scratch and configured at runtime.

Build and Deploy: 1.5X
- Because of automation, parallelism, and orchestration, an entire working environment can be holistically deployed in working condition.

Development: 10X
- 1X copy each brownfield VM and configure how to provision a new VM
- 4X post provision configuration for each new VM: load balancer, web, and database server (hard coded).
- Test that a deployment works and polish if needed.
- 3.5X rework web server to use variable from database address and change to a web-tier population; rework load balancer to use variable from web tier addresses for configuration.
- Test that deployment works and polish if needed.
- 1.5X scale-out or scale-in web-tier configuration change for load balancer.
- Test that deployment works and polish if needed.

MTTR: 1.5X
- Repairs change from restoral to redeployment

Benefits:
- Push-button, fleet deployment with huge savings in deployment time
- Scale out web-tier with automated load balancer reconfiguration: a runbook for lifecycle operation is a push-button change control.

Disadvantages:
- Requires a human to initiate a new deployment, scale-out the web tier, or to decommission the deployment.
- Can't be delegated to end-users for self-service unless basic governance is established: identity, potentially multi-tenant Role-Based Access Controls (RBAC), resource quotas (for CPU, memory, storage), operations audit logs, alerts, and reporting facilities required.
- When happy with this new deployment, adding a DNS round robin record with the old and new load balancers could eliminate that single point of failure from the design, but it is a manual operation.

CONFIGURATION MANAGEMENT: SHELL SCRIPTING ON STEROIDS

This step may be considered optional, but mastering it can revolutionize the approach to adopting new workloads with automation.

Cloning the blueprint from stage 5, one can refactor Powershell scripting to use configuration management (CM) instead. Briefly, CM is a category of software that can be considered the next generation of scripting. It may also be beneficial to bootstrap from community contributions for web and database, potentially allowing you to change operating system versions, which may help with your workload portability. Many provisioning tools can orchestrate configuration management on each VM guest--client-only CM has been very popular for adopting this new ability. The more advanced CM systems support client-server models with change management database (CMDB) features, secure orchestration of secrets, fleet management, and more.

Blueprints can easily bootstrap, configure, and run a configuration management client tool. In fact, you can mix and match your old procedures and new CM procedures, or even multiple CM systems, inside a blueprint. This flexibility is crucial for expressing your work using your current skill sets alongside new facilities, while also leaving open the option to refactor for future efficiencies. Also, given that different teams typically adopt different tools, your system should be able to accommodate this heterogeneity through orchestration--bringing all of the knowledge in your organization under one roof to deliver business with a single click.

Perhaps the most interesting benefit of CM is that after the initial provisioning and execution, periodic rerunning of CM on each VM can identify and (if desired) remediate configuration drift back to the desired state. This capacity constitutes one aspect of self-healing infrastructure and is an essential tool for establishing on-going security of your environments. (Some CM tooling can work in a client-only mode; we assume this capacity below for simplicity of adoption.)

The configuration management communities offer official and community-supported workload automation, which can accelerate new workload adoption. Once becoming comfortable with the benefits of configuration management, CM support may become a new requirement or qualification for any new vendor, workload, or solution. CM can also expand to large client-server systems to manage large fleets of datacenters and VMs.

The configuration management option has begun to wane, however, with the onset of Docker containers and Kubernetes. These technologies have led application architecture towards immutable infrastructure artifacts, which are built and orchestrated with a minimum of runtime configuration--in contrast to entirely synthesized configuration during provisioning time (as seen in this and the previous stage). These more lightweight options optimize time, storage, and compute, so the next stage begins down that path.

Design:

- A copy of stage 5, refactoring Powershell to configuration management.
- 5 servers, synthesized from scratch and configured at runtime.

Build and Deploy: 1.5X

- Because of automation, parallelism, and orchestration an entire working environment can be holistically deployed in working condition.

Development: 11X

- 4X: survey the market, identify, and install a pilot CM system.
  – Test that a deployment works and polish if needed.
- 3X: identify CM community/official support for load balancer, web, and database.
- 3X: bootstrap CM and then refactor Powershell to load balancer, web, and database CM or reduce Powershell to leverage blueprint variables and configure inputs to CM.
  – Test that a deployment works and polish if needed.
- 1X: schedule periodic CM runs for configuration drift detection

MTTR: 1.5X

- Repairs change to manual redeployment or potential CM remediation.

Benefits:

- Increased security for configuration drift detection
- Potentially easier to change VM operating systems for upgrades
- Potential CM automated remediation of workloads

Disadvantages:

- Requires a human to initiate a new deployment, scale-out web tier, or scale-out web-tier, or to decommission the deployment.
- Can not be delegated to end-users for self-service unless basic governance is established: Identity, potentially multi-tenant Role-Based Access Controls, resource quotas (for CPU, Memory, Storage), operations audit logs, alerts, and reporting facilities required -- requires larger client-server CM investment to potentially cover some or all of these requirements.

07

CREATING BUILD-TIME INFRASTRUCTURE ARTIFACTS

This stage requires adopting a build system and creating a build script. There can be no turning back from being an infrastructure developer once mastering shell scripting, Infrastructure as Code, or CM! You will adopt more of the software engineering discipline in this stage to build your infrastructure and treat it like an application: easy to build, test, and deploy.

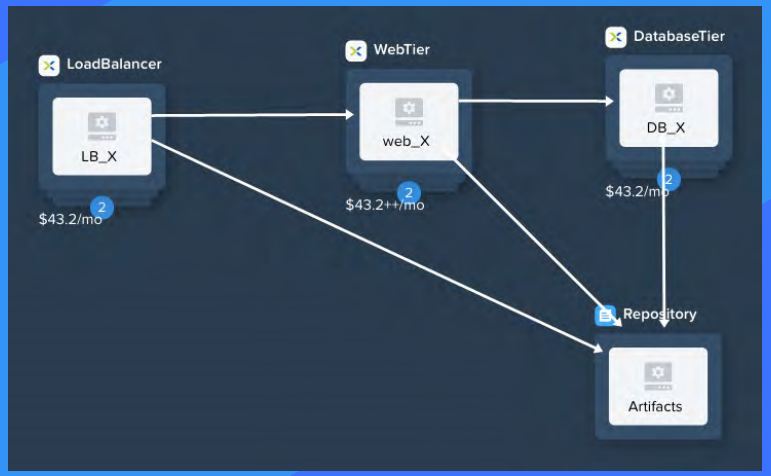
The goal is to manually build and orchestrate as much as possible into a VM template or operating system image. By doing so, you can regularly build in operating system, application server, and software library updates and security patches. These are infrastructure artifacts that have been built, distributed to some file repository, and can be provisioned quickly with a minimum of runtime configuration, greatly speeding up development, test, and production environments, as well as backup and scale-in and scale-out operations.

Another refinement at this stage is to move from VMs to Docker containers and from OS builds to application builds of software packages. The Docker container model shortens build and deployment times (the latter down to one second), which fuels a rapid feedback loop for development and testing. With containers you can also reduce the build to include only the application and its dependencies, because the operating system and most OS utilities aren't required to run the application. Finally, you can redesign the build artifact container to be read-only (with environment variables providing the configuration information), which lets you build the container once and then re-use it for development, testing, staging, and production environments.

Read-only (immutable) artifacts send logs, state, and transactions over the network to other services, making it stateless and easy to deploy and redeploy with minimal interruption. Reusing the build artifact minimizes the deltas between environments, preventing the historical need for operating procedure changes in different deployments, and in turn driving out operational risk.

It's worth noting that Windows containers could work well on the web application tier. However, because Windows containers are still in an early stage of general acceptance, we do not recommend it here.

Both of the preceding options move from runtime configuration management to build-time infrastructure artifacts, a crucial step towards the next stage of software engineering adoption--test automation and faster deployment times.



Design:
- Refactoring of the blueprint in stage 6 to use a build artifact management.

Build and Deploy: 0.75X
- Evaluate the beginning of runtime configuration in the blueprint for the web, database, and load balancer, identifying common procedures that are performed on every server. These procedures typically represent best practices, such as: updating the operating system, installing agents, and running scans.
- Refactoring these common operations into a standard OS image or VM template (sometimes called a golden image) can reduce the amount of runtime configuration, allowing faster deployment.
- Estimating an average 25% savings, but it can be as much as 90% in some cases.

Development: 3X
- 1.5X Create a simple file share or artifact repository to store a base OS image
- 1.5X Manually create a golden image every week; publish it to the repository; update the blueprint to use this new golden image.

MTTR: 1X
- Remediation can be as simple as a new deployment and cutover.

Benefits:
- Deployments are faster and push button.
- Easy to update base operating system and security patches and roll out with next deployment.
- Easy to test new deployments and operating models by adjusting the blueprints and rebuilding with a push button.

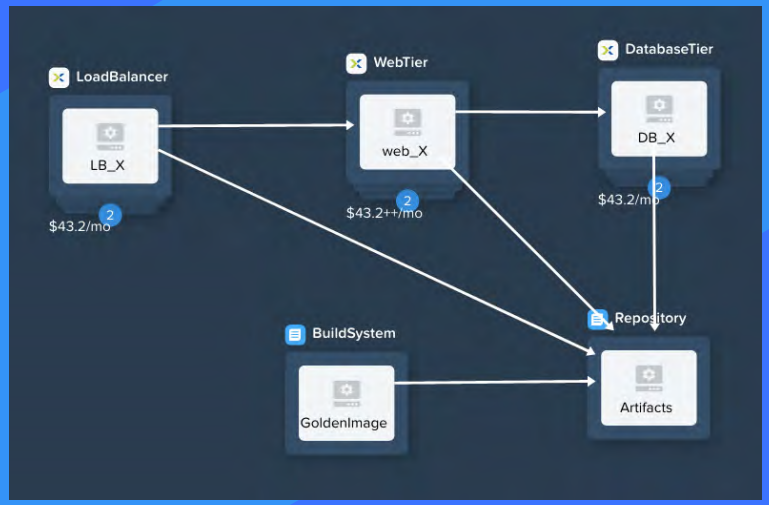
Disadvantages:
- Further build time optimization is possible, see the next stage.
- Not containerized yet for further deployment acceleration.

CONTINUOUS INFRASTRUCTURE INTEGRATION, DELIVERY, DEPLOYMENT, AND OPERATIONS

In this step we discuss how you can use blueprints and test automation to achieve continuous integration, delivery, deployment, and operations. The blueprint from stage 7 can be cloned and refactored to accept runtime arguments to the VM or container artifacts. A build system script or build job can call the blueprint to orchestrate the new artifacts for Continuous Integration (CI) of these artifacts, verifying that they can be deployed and pass the build. Typically, a developer thinks of the build system as the source of truth for how to deploy an application, but complex deployments require both an orchestrator and operational testing of the application lifecycle.

You can also test application integration and, if successful, promote the artifacts to an environment, thereby achieving Continuous Delivery. Finally, with extensive test automation, the system can automatically repeat the deployment and promote the candidate artifacts to run in production to achieve Continuous Deployment.

Blueprints can host many different sorts of operations, including testing application integration from an operational as well as a functional standpoint. For example, continuous operations requires capacities such as automated problem remediation. You can use the blueprints to test for this capacity by injecting infrastructure faults or failures in a safe and controlled manner. The goal is to ensure the availability of a delegated, lights-out, self-healing application experience. In contrast to traditional operations, which require humans touching gear in the datacenter to remediate, you can secure zero-touch, guaranteed uptime and reliable operational maintenance.



Design:

- Leveraging a build system to create and store a build each server artifact (LB, DB, Web) and call a new blueprint to use these artifacts.

Build and Deploy: 0.25X

- Refactoring installation and general configuration operations into a standard OS image or VM template (sometimes called a Golden Image) per server artifact can reduce the amount of final run-time configuration, allowing faster deployment.
- Estimating an average 75% savings, but it can be as much as 99% in some cases.

Development: 13X

- 2X stand up and pilot a simple build system or service (such as Jenkins)
- 2X create and configure a simple build job named "Golden Image"
  - Download, patch the latest operating system, apply any other common operations and facilities to a VM, and test.
  - Snapshot the VM and publish the VM artifact to the repository and test.
    - This step can be skipped if you simply use the build system as your artifact repository, but that will incur storage overhead unless you regularly purge old builds
- 2X leverage a source code repository (such as Git, Github, or Gitlab) to store your "Golden Image" build job code and optionally use a web hook to trigger the build job for every code commit change and test.
- 2X extend the "Golden Image" build job to call a modified "Web Application-Golden Image" blueprint with the latest build artifact location:
  - Clone the existing Web Application blueprint and rename it to "Web Application-Golden Image"
  - Refactor the blueprint to accept the build artifact location as a runtime argument.
  - Update the "Golden Image" build job to call the "Web Application-Golden Image" blueprint and test.
- This completes the updated VM golden image and blueprint consumption exercise to achieve a continuous integration and delivery scenario.
  - Further optimization for deployments can be made by making specific DB, Web, and LB golden images; the next steps enable this optimization.
- 5X create a parallel CI/CD pipeline with optimized deployment utilizing one meta-build job for three artifacts and a modified blueprint to consume those three artifacts:
  - Clone the existing "Web Application-Golden Image" blueprint, rename it to "Web Application-Multiple Golden Images" and modify the blueprint:
  - Modify it to accept three build artifact locations as run-time inputs: one for the web, database, and load balancer.
  - In the following steps, pull some of the installation procedures out of the build artifact blueprint and moving those procedures to each build individual web, database, and load balancer artifact build job to shift work from run-time deployment to build-time.
  - While the following appears to be three times the work, it isn't because it refactors existing work already created in the blueprint.
  - The new "Web Application-Multiple Golden Images" blueprint will retain any final configuration stages and starting of the server that cannot exist at build time.
- Localize a build job for the web server artifact:
  - Copy the OS prototype build job and rename it to include "web server artifact."
  - Add installation and generic configuration of the web server by moving some of the functionality from the "Web Application-Golden Image" blueprint to the build job.
  - Adjust publishing the web server artifact to a repository and test.

- Localize a build job for the database server:
  - Copy the OS prototype build job and rename it to include "database server artifact."
  - Add installation and generic configuration of the database server by moving some of the functionality from the "Web Application-Golden Image" blueprint to the build job.
  - Adjust publishing the database server artifact to a repository.

- Localize a build job job for the load balancer server:
  - Copy the OS prototype build job and rename it to include "load balancer server artifact."
  - Add installation and generic configuration of the load balancer server by moving some of the functionality from the "Web Application-Golden Image" blueprint to the build job.
  - Adjust publishing the load balancer server artifact to a repository.

- Create a new build job called "web app artifact build and deploy:"
  - This will be a meta build job; it will call each of the three new artifact build jobs for DB, Web, and Load Balancer, capture each of the artifact published locations, and call the "web application with build artifacts" blueprint with the three artifact locations.
    - In this manner, one can update each artifact build job independently without tampering with the others.
    - Alternatively, the three artifact blueprints could be composed into this as a monolithic build job, but that would not be a modular approach, so it is not advised as a best practice.
  - Test and fix the build job and blueprint until everything works properly!

MTTR: 0.5X

- Repair and updates can be a redeployment of the web and load balancer = .25X
- Database repair and updates require longer times and procedures outside the scope of this exercise, so the MTTR average is raised to 0.5X

Benefits:

- Multiple pipelines of progressive continuous integration and delivery scenarios which decrease deployment time and MTTR
- Continuous integration of the Golden Image with a build job
- Continuous delivery of the Golden Image with a blueprint for testing
- Composure of Golden Image into specific DB, Web, LB build artifacts for faster run-time deployment and configuration.
- Meta-build job of all three golden image artifacts enables decomposure into specific artifact jobs that can iterate independently of each other.
- Build jobs can be triggered by build server users ad-hoc or by check-in to update a job.

Disadvantages:

- Not containerized yet for further deployment acceleration.

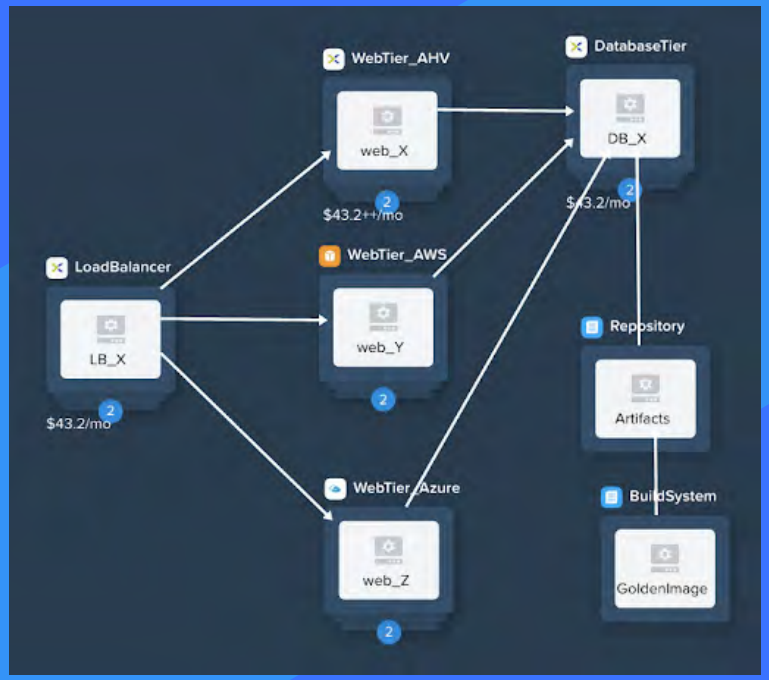
09

ENTERPRISE, HYBRID, AND MULTICLOUD DEPLOYMENTS

The next stage is to expand deployments to multiple providers. This is yet another refinement of the blueprint to ensure that one can deploy and blend workloads across on-prem and off-prem infrastructure offerings, achieving a hybrid cloud stance. Most importantly, the business must retain a consistent governance and operational model so that workloads can meet fiscal and customer performance service level objectives, no matter where the workload lands.

You can clone and refactor the blueprint from stage 8 for parallel deployments to multiple hypervisors, public clouds, and datacenters. You can also schedule and orchestrate a mixture of VMs and containers into Kubernetes pods for hybrid deployments across multiple clouds, supporting teams at different stages of container maturity. This flexibility allows all of the infrastructure market to compete for your business service level agreements. Freedom of choice among vendors empowers you to not only select the best products and services, but the optimal fiscal and operational models.

This was a key finding from **Accelerate: State of DevOps 2019**, where "the highest performing teams were 24 times more likely than low performers to execute on all five capabilities of cloud computing."



Design:

- A copy of the "Web Application-Golden Image" blueprint clones the web tier and localizes it for a second provider.
- If the current load balancer can route over the network to the second provider web-tier instances, it can load balance between both infrastructure providers.

Build and Deploy: 0.25X

- Parallel deployments of golden images across multiple providers can preserve the optimized deployment time from stage 8.

Development: 2X

- Clone the "Web Application-Golden Image" blueprint and rename it "Web Application-Golden Image-Multiple Providers"
- Copy the web tier in the blueprint, localize it to provision an equivalent golden image as a VM in the second provider, and ensure the VM instances are publically available so that the load balancer in the original provider can route over the network to these new provider VM instances.
- Update the load balancer with the network address of these new VM instances and test.

MTTR: 0.25X

- Database repair still keeps MTTR average at 0.25X because it hasn't been redesigned.

Benefits:

- Designing the web tier for failure allows it to be distributed among multiple infrastructure providers, providing better resilience and guarding against any single pet provider's inevitable failure.
- A new infrastructure provider can be tested easily once a blueprint deployment scenario has been enabled.
  - This can be repeated for additional providers, datacenters, availability zones, and competitors.

Disadvantages:

- Health checks of web-tier instances should be automated through monitoring for removal and restoral operations to make this a self-healing, scalable web-tier.
- Advanced operations and configuration for replication and failure+restoral reconciliation between multiple database instances is outside the scope of this exercise, but it is another important design consideration.

HEURISTIC-DRIVEN OPERATIONS

The final refinement of the blueprint from stage 9 would be to clone and augment it with global load balancing among the different providers. Adding monitoring, metrics, and logs allows one to heuristically trigger autoscaling workloads with load-balancing operations. Zero-click operations give you a self-healing, anti-fragile business with "continuous operations." At this stage you have global application deployment and load balancing, driven by business-focused key performance indicators, service level agreements, and operational health that scale hybrid workload populations up and down across the world automatically. Your applications are highly available, performant, and close to your customers when they need them.

Design:

- A copy of "Web Application-Golden Image-Multiple Providers" blueprint will be extended with monitors to enable advanced operating modes of autoscaling and healing.

Build and Deploy: 0.25X

- Parallel deployments of golden images across multiple providers can preserve the optimized deployment time from stage #8

Development: 12X

- Clone the "Web Application-Golden Image-Multiple Providers" blueprint and name it "Web Application-Golden Image-Multiple Providers-Autoscale" blueprint
- 2X: Add Load Balancer service configuration to inspect the current population of the web tier VMs (across all providers) with a basic monitor to remove failed instances.
  - HTTPS response on the expected port is a typical health monitor available in most load balancers.
- 3X: Pilot a basic monitoring server or SaaS with an API for creating and deleting health checks of CPU load on a VM.
- 7X: Update the "Web Application-Golden Image-Multiple Providers-Autoscale" blueprint operations:
  - 1X Create and configure a generic CPU load agent in the golden image build job.
  - 1X Update the blueprint with monitoring service end-point and credentials, configure a runtime update to the configuration for the monitoring service for CPU load on web-tier VMs for each provider.
  - 2X Create a monitoring policy to contact the appropriate provider web-tier operation in the blueprint to scale-out the web-tier by one VM, never exceeding 4 VMs (or your business policy), when any web-tier VM in the provider reaches > 80% load for more than two cycles.
  - 1X Create a monitoring policy to contact the appropriate provider web-tier operation in the blueprint to scale-in the web-tier by one VM, always leaving at least one VM, when any web-tier VM in the provider reaches > 80% load for more than two cycles.
  - 2X Create an operational procedure in the blueprint to generate 100% CPU load for 3 health check periods a web-tier VM for each provider to test scale-out and scale-in.

MTTR: 0.1X

- Repair can be removing failed web instances from the load balancer, which can be automated with periodic, basic health checks. The period could be 0.01X for web health check and MTTR.
- Database repair still keeps MTTR average at 0.25X because it hasn't been redesigned.

Benefits:

- Auto scaling and healing web tier driven by health monitors across multiple providers.

Disadvantages:

- To prevent any pet load balancer or pet infrastructure provider instance failure, it is still necessary to provide global load balancing to "herd" the infrastructure providers.

READY TO BECOME AN INFRASTRUCTURE DEVELOPER?

This ebook has shown the successive stages of a relentless pursuit to improve application architecture, infrastructure architecture, and operational excellence --all in the name of delivering business value. The path toward DevOps and agile practices can be arduous, as it requires removing traditional operational designs and constraints. Even so, it delivers a wide range of benefits at every step of the way. By the time you reach the 10th step, however, you will have achieved the following improvements:

1. Decreased deployment time:
- Reduced to one quarter (0.25X) of the initial effort in stage 1
- Deployment moved from VMs to artifacts to become simpler, deterministic, and reproducible operations
- Avoided the lift and shift trap!

2. Improved uptime:
- Allowing maintenance and tolerating some failures
- Distributing risk and removing single points

3. Improved time to market:
- Continuous delivery through automation removes human effort and errors, while improving agility
- Initial delivery can be to internal customers, such as for development, testing, and staging environments
- Automated guardrails can enable continuous deployment to production and external customers

4. Mean Time to Repair (MTTR):
- Decreased to nearly nothing; one fifth (0.1X) of the initial MTTR (0.5) in stage 1
- Quickly remedy any mistakes found in production by deploying previous artifact versions

5. Global load-balanced infrastructure:

- Eliminate every single point of failure, moving to an active-active operational model for business uptime and continuance in case of disaster
- This strategic advantage enables all vendors to compete for your business service-level agreements and key performance indicators
- A choice of provider models can enable fiscal blended models of capital and operational expenditure

6. Business key performance indicators drive automated operations:

- Dynamically scale capacity up or down, and in or out, based upon measured customer experience and demand
- Employees drive better automation and predictive intelligence into business operations, away from hands-on remediation and reactive operations to emergencies
- Improvements can be tested on a small sampling of the customer population before full roll-out.

This table lets you see at a glance the changes in development time at each stage with corresponding deployment and MTTR benefits. What's clear is that the transition from a traditional, hand-built, monolithic single environment to ephemeral, on-demand, hybrid, multicloud continuous application deployments constitutes a radical transformation that facilitates rapid innovation and delivery of customer value.

STAGE	DEPLOY AND DEVELOPMENT		MTTR	SECTION TITLE
1	1		0.5	The Monolithic in Production
2	1.5		0.75	Separate, Monolilthic Functions
3	6		3	Load Balancing for Uptime
4	1	4	0.5	Application Automation and Orchestration
5	1.5	10	1.5	Synthesis with Scripting: Push Button Environments
6	1.5	11	1.5	Configuration Management: Shell Scripting on Steroids
7	0.75	3	1	Creating Build-Time Infrastructure Artifacts
8	0.25	13	0.5	Continuous Infrastructure Integration, Delivery, Deployment, and Operations
9	0.25	2	0.25	Enterprise, Hybrid, and Multicloud Deployments
10	0.25	12	0.1	Heuristic-Driven Operations

In the end, we all want our business to work as aggressively and efficiently as the webscale giants, but the prescription has been unclear. In this eBook, we have presented concrete steps to start refactoring your business, applications, operations, and, indirectly, your culture to move toward a data-driven, experimental way of growing and improving your business. When done right, DevOps delivers continuous innovation and operations at scale, anywhere in the world.

If you are just beginning your DevOps journey, visit Nutanix.com to learn about how you can modernize your datacenter with revolutionary technologies like hyperconverged infrastructure (HCI). The HCI-based Nutanix Enterprise Cloud eliminates infrastructure silos and frees you from relying on tightly coupled, hand-built "pet" servers maintained by "pet" IT specialists. Nutanix web-scale architecture delivers always-on availability and resilience with a distributed "herd" of servers.

And it's so easy to use that IT generalists can conduct formerly time-consuming and complex operations like managing, scaling, automating, and troubleshooting IT infrastructure with a single-click.

Nutanix HCI provides a foundation to modernize self-service, on-demand, global delivery of applications on multiple invisible compute providers anywhere. Nutanix is designed for ephemeral, hybrid, multicloud IT, which simultaneously fosters vendor competition for your business across any consumption model, while also eliminating single points of failure. Most importantly, IT can now easily rise above infrastructure to deliver true continuity for the entire business stack, achieving much higher efficiencies and better outcomes than ever before.

If you're already on the DevOps path toward non-disruptive, continuous innovation and operations, then check out Nutanix.dev, which includes an array of resources, ranging from labs, working scripts and example apps, official documentation for Nutanix APIs, developer community blogs, events, and more.

NUTANIX™
YOUR ENTERPRISE CLOUD