WHITE PAPER–MAY 2020

Tackle Application Modernization in Days and Weeks, Not Months and Years

Table of contents

Why you need application modernization

Most developers work on existing applications: products and services that have been built, maintained and updated over long periods of time. Normally, these apps exist as a web of tightly coupled, sparsely documented systems.

Over time, large organizations develop layers of manual processes designed to minimize risk and ensure compliance. As a result, software releases are often infrequent, high-ceremony events that require heroism and brute force. This bleak status quo is untenable for businesses that want to compete and win in the digital age. The question you need to answer is simply: “How can I refactor my most important apps, so I can get new features to production faster?”

VMware Tanzu Labs has answered this question for hundreds of enterprises. The foundation of this success: a modernization methodology that leverages cloud native patterns and continuous delivery (CD) automation across your existing application portfolio.

The benefits of CD are clear (and outside the scope of this paper), but getting there is a universal challenge. We take an iterative approach to modernization by starting in a small, focused way. We help you realize incremental time, cost and operational efficiencies while improving security, resilience and compliance. We pair with you to update and automate processes incrementally, while transforming your applications. This yields a win-win for you and your teams: It reduces the pain of releasing updates more often, while meeting uptime, security and compliance requirements. Let’s take a closer look at this methodology.

Our application modernization approach

There are different reasons and many ways to start a modernization journey. Two rationales are most common:

1. Portfolio transformation – Focused on broad and strategic efforts to migrate, refactor and transform an existing portfolio to cloud-based technologies. Enterprises well on their way to cloud still find it very challenging to realize desired outcomes as they continue to fight against technical debt and organizational inertia. Stakeholders want to move quickly, retire heritage assets, and capture time and cost-centric outcomes across the portfolio.
2. System modernization – Relates to an existing system of systems that’s business-critical, expensive to update, technically complex and under active development. New feature demand or business direction prolongs delivery time and increases cost while magnifying architectural complexity. Cloud technology and cloud native architectural patterns provide an opportunity to solve problems.

Portfolio transformation is challenging from a technical, business and human perspective. It requires your teams to develop new skills, while making major changes to architecture and software development lifecycle (SDLC) processes. Change can often result in IT reorganization, a frightening prospect. To be successful, organizations must perform a delicate balancing act: Dedicate their most talented people to the effort, while ensuring that the entire portfolio continues to operate.

The keys to system modernization success lie in understanding the root cause (e.g., bloated, monolithic code base) before a solution is determined. Sometimes, a technology change is all that is needed. But more often, the problem is complex and requires strategic thinking, planning and iteration. Either way, it’s critical to quickly get to the heart of the concern. Avoid the temptation of fixing symptoms (e.g., UI responsiveness), even though those are easier to see.

Tanzu Labs has helped hundreds of organizations on this journey. Our approach defines incremental steps that gradually increase the cloud maturity of existing systems, the automation in your SDLC and the knowledge of your team.

Before we get into detail, let's define some vocabulary and core tenants.

Types of application modernization

The word "modernization" can mean different things to different people. We see modernization as an approach that improves an existing piece of software. We define improvement as making updates that align with IT and business outcomes. We also have seen technology companies, analysts and system integrators articulate options for this type of work, including, for example, lift and shift (focused on workload containerization). However, we like to keep it simple and, when possible, avoid buzzword bingo. As such, we believe there are two general ways to modernize existing systems:

Replatform – Targets self-contained applications or sub-system modules, and uses minimal effort to run executables in *OCI*-compliant containers. Outcomes are IT-focused and often related to goals such as higher operational efficiency (e.g., fewer people to run more containers) and better infrastructure density (e.g., multiple containers per virtual machine). Effort is low.

Refactor – Code is converted to run in a *15-factor-compliant way* by moving thin slices of functionality to cloud native patterns. The goal: Deliver fast, iterative results while providing interoperability features for continuous operation with the old system. This process starts with rapid investigation of an existing system to find root problems and opportunities. The output of this exercise flows into the next step: iterative work to make improvements. Effort can be moderate to high depending on slice definition, intended outcomes and degree of technical debt. The work is crucial to unlocking the desired business outcomes.

These types are not mutually exclusive. Portfolio transformation typically starts with replatforming efforts that emphasize SDLC process and skill improvements, before getting into larger, more complex modernization work. Modernization projects, especially those that span multiple systems, will likely also include replatforming. More than anything, it's about outcomes and getting there as quickly as possible in a sustainable way.

Tenets of application modernization

These are the four elements of a successful application modernization initiative:

1. Start small – Even if your portfolio contains thousands of apps, start with a single business unit and a handful of applications that matter.
2. Automate everything – Use test-driven development, continuous integration and continuous deployment to reduce manual process time and SDLC cost.
3. Learn by doing – Inform strategy and build new skills through hands-on effort, rapid feedback, measuring results and by creating a cookbook of patterns as you go.
4. Break things down – Iterate quickly and continually on thin slices of complex systems.

We emphasize these tenants to quickly focus on the right things and deliver impactful, iterative results.

Prerequisites for the application modernization journey

Starting the work requires a list of viable apps to migrate, organizational commitment and the right people. It's critical to start small; initial efforts will be intense and learning-oriented.

A list of viable application candidates

Start portfolio transformation with a set of custom apps that have business relevance and are active use.

Organizational commitment

A motivated business unit with leadership committed to cloud and a willingness to invest time and dollars in transformation.

The right people

A small team of people that understand the application domain(s), who are made available to work on the initiative in a dedicated way.

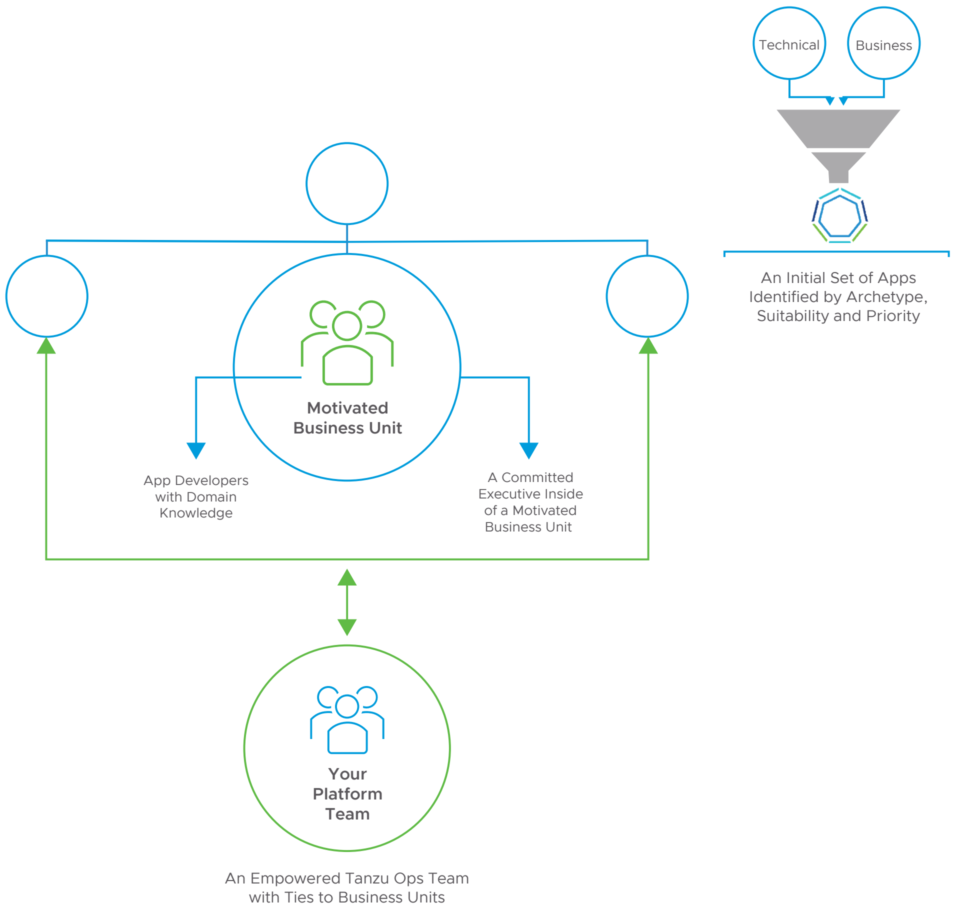

Technical
Business
An Initial Set of Apps Identified by Archetype, Suitability and Priority
Motivated Business Unit
App Developers with Domain Knowledge
A Committed Executive Inside of a Motivated Business Unit
Your Platform Team
An Empowered Tanzu Ops Team with Ties to Business Units

FIGURE 1: Apps, commitment and people.

Organizational commitment

The application modernization journey must begin somewhere, usually within a business unit. It's critical that local leadership be motivated and willing to invest time and money into the transformation. A series of small successes are needed before application modernization can spread across the larger organization.

Sponsorship is usually a CIO or senior executive responsible for transformation within the target business unit and who controls a relevant budget. The sponsor must have the political capital to unblock existing policy and process, as well as other obstacles standing in the way of the app transformation team.

The other aspect of organizational commitment: mission and goals. A broad North Star goal and tangible, near-term objectives are critical to defining the effort. This will determine funding, measurement and intended duration. These things may (and often should) evolve. But it's important to both motivate and measure people against what success looks like.

A list of viable application candidates

Portfolio transformation starts with a basic understanding of technical suitability. This is how you'll choose a first set of applications to be refactored. Applications will be selected based on a combination of Tanzu Labs technical, business and organizational factors. Selection criteria will be codified into a decisioning model, discussed in *Start and scale portfolio transformation*. That said, establish a rough sense of apps that matter from a business perspective. The ideal candidate apps should run without high cost or complexity on the chosen cloud platform.

The right people (aka a seed team)

Our most successful Tanzu customers have assembled a small team of employees to drive the application modernization initiative. The dedicated team includes a product owner and (ideally) developers familiar with the initial candidate applications to be transformed. The team should also have experience (and/or training) with the target platform and cloud native architecture principles. And, finally, it's critical that the seed team be empowered by leadership to make decisions without lengthy process ticketing or sign-offs.

CORE ROLES WITHIN SMALL CROSS-FUNCTIONAL TEAMS	
Product owner	Represent business interests through backlog prioritization and internal coordination to unblock encountered issues by the team to ensure maximum project velocity.
Project anchor	Hands-on technical leaders who pair with product owners on backlog concerns, guide technical practices, oversee quality and do technical work.
Developer	Skilled architect/developers who know the existing app and underpinning stacks being worked on as they grow cloud native skills and experience by doing the work.

Give your applications a modern runtime with a production-grade platform infrastructure

If possible, a production-grade platform should be available as a prerequisite. It should be operated by a dedicated platform product team. Expect frequent interaction with the platform group, as this team will assist with configuration updates (networking, storage, logging, etc.) to fit application requirements.

Start and scale portfolio transformation

Effective decision-making is critical. We recommend establishing an *ongoing work stream*, running parallel to a technical track of modernizing applications. You'll need to designate someone to own identifying business and technical criteria, governance and measurement. You'll also need this person to wade through the inter-company processes (funding, people, incentive structure, etc.) needed to facilitate the work. Ideally, this effort yields a set of assumptions used to make decisions around technical suitability, business characteristics and organizational factors—key inputs for prioritization.

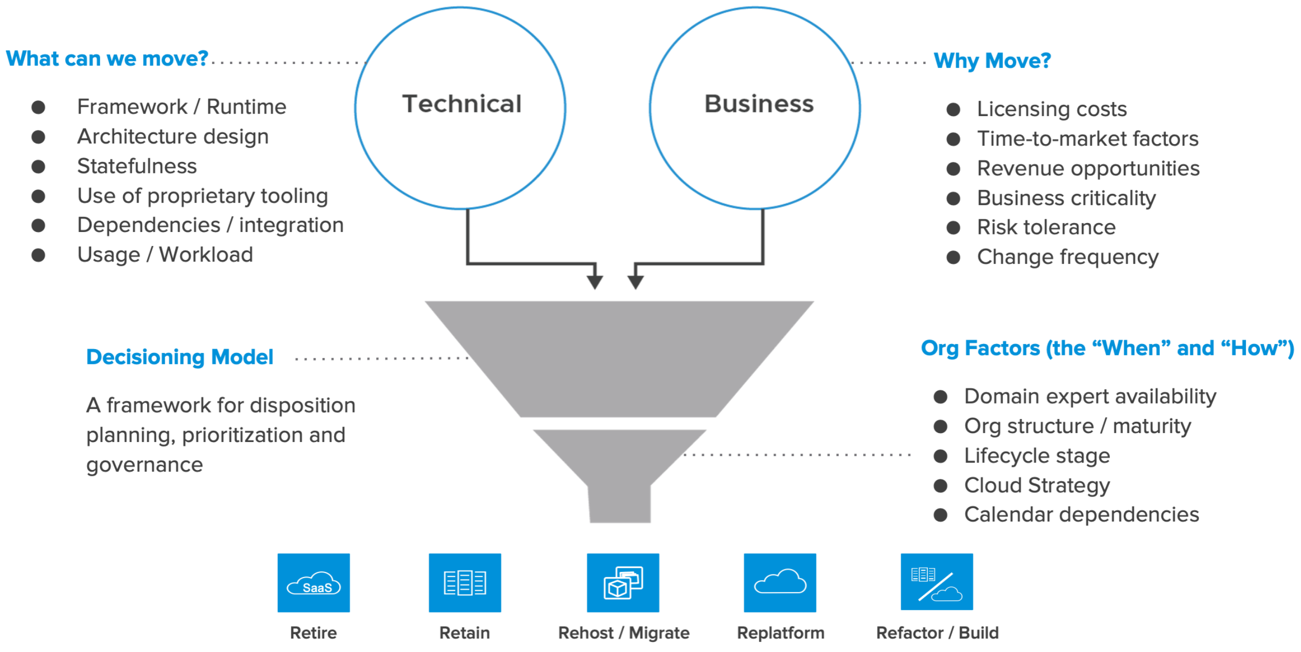


FIGURE 2: Modernization decision-making fundamentals.

We measure technical suitability using *15 technical factors*, an update to the *12-factor app* manifesto published by Heroku. Compliance with all 15 factors generally means your app is cloud native, will run on a modern cloud platform and take full advantage of elastic cloud features, such as auto-scale. In our experience, most applications in your portfolio don't require the full cloud native treatment. Instead, they can be replatformed to achieve IT outcomes with minimal time and cost. Full cloud native compliance is best achieved through new greenfield initiatives or via modernization projects having a supporting business case.

How to select an initial set of applications

Select an initial sampling of apps representative of the broader portfolio in terms of technical design or archetype. The perfect apps would be of lower technical complexity. This allows the new team to learn, earn some wins and spend their time on process improvements (instead of tedious code refactoring). Companies in regulated industries should ensure that these apps reflect common policy, security and compliance constraints. And remember to look at data—apps tightly coupled to a shared monolithic database are harder to refactor due to collateral impact on other connected systems.

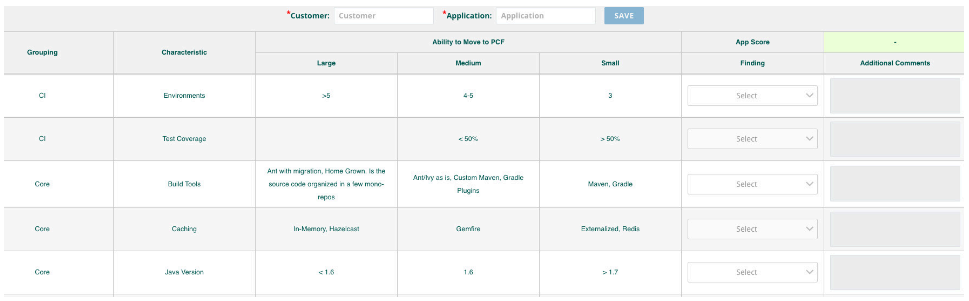
Analysis: Prioritize modernization across your app estate

Prioritization requires a clear sense of business and organization factors that signal the why, how and when of modernization. Defining these factors is subjective and will be far from perfect. What's most important is to use ongoing feedback to sharpen your selection criterion over time as the initiative grows.

Tanzu Labs generally uses a simple 2x2 matrix and facilitated conversation to quickly determine prioritization and to flag questions, assumptions and risks. The X and Y axes often frame complexity and value from low to high to categorize apps in scope. When a large number of apps are plotted, they tend to cluster together and it's useful to create a child matrix for deeper probing. This exercise will help establish a general consensus around which apps balance technical suitability with business impact. From there, more analysis is required in context of apps deemed highest priority.

Analysis: Quickly assess technical suitability

SNAP (snap not analysis paralysis) is an approach used to evaluate the technical suitability of application candidates when source code is unavailable. SNAP covers app usage, architecture and configuration. It can be done manually in 10-15 minutes on a small set of apps. Tanzu Labs conducts this exercise using facilitated conversation and stickies, or by using a web application (see Figure 3).

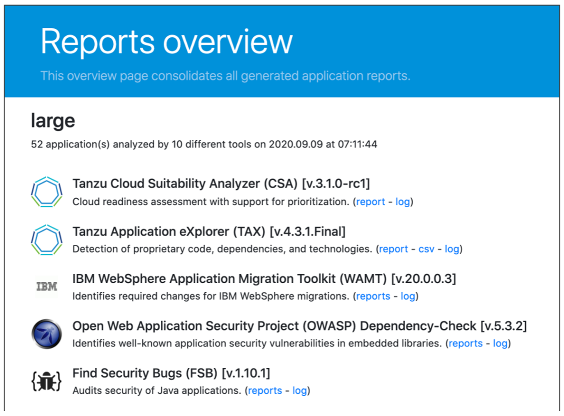


Grouping	Characteristic	Ability to Move to PCF			App Score	-
		Large	Medium	Small	Finding	Additional Comments
CI	Environments	>5	4-5	3	Select	
CI	Test Coverage		< 50%	> 50%	Select	
Core	Build Tools	Ant with migration, Home Grown. Is the source code organized in a few mono-repos	Ant/Ivy as is, Custom Maven, Gradle Plugins	Maven, Gradle	Select	
Core	Caching	In-Memory, Hazelcast	Gemfire	Externalized, Redis	Select	
Core	Java Version	< 1.6	1.6	> 1.7	Select	

FIGURE 3: Web version of SNAP.

Analysis: Automated source code analysis

Application code has a story to tell and, when available, you can perform SNAP automatically on hundreds or even thousands of apps using our industry-leading automated assessment tooling combined with leading open-source frameworks. We can provide insight in terms of cloud suitability, security, language, licensing and quality.



Reports overview

This overview page consolidates all generated application reports.

large

52 application(s) analyzed by 10 different tools on 2020.09.09 at 07:11:44

Tanzu Cloud Suitability Analyzer (CSA) [v.3.1.0-rc1]
Cloud readiness assessment with support for prioritization. (report - log)

Tanzu Application eXplorer (TAX) [v.4.3.1.Final]
Detection of proprietary code, dependencies, and technologies. (report - csv - log)

IBM WebSphere Application Migration Toolkit (WAMT) [v.20.0.0.3]
Identifies required changes for IBM WebSphere migrations. (reports - log)

Open Web Application Security Project (OWASP) Dependency-Check [v.5.3.2]
Identifies well-known application security vulnerabilities in embedded libraries. (reports - log)

Find Security Bugs (FSB) [v.1.10.1]
Audits security of Java applications. (reports - log)

FIGURE 4: VMware Tanzu report generator.

Define the path to production

Chances are your existing SDLC processes have been in place for a long time. They likely involve diverse sets of people and approved tooling standards. Further, your incumbent practices are likely organized around important policies and regulations. Getting source code from a developer's laptop into production requires manual effort and time. Cloud platforms, when combined with CI tooling and CD techniques, can automate much of the toil, and significantly accelerate the frequency of production deployments.

To refactor your path to production, you should first baseline what exists today. Tanzu Labs uses in-person techniques such as *event storming* to discuss, probe and plot the sequence of activities involved in releasing software. This can often be done in a couple of hours. The end result: a rough approximation of process flow, opportunities and areas of risk. Then, we map the findings and translate our notes into a spreadsheet and continue to quantify cycle time and cost.

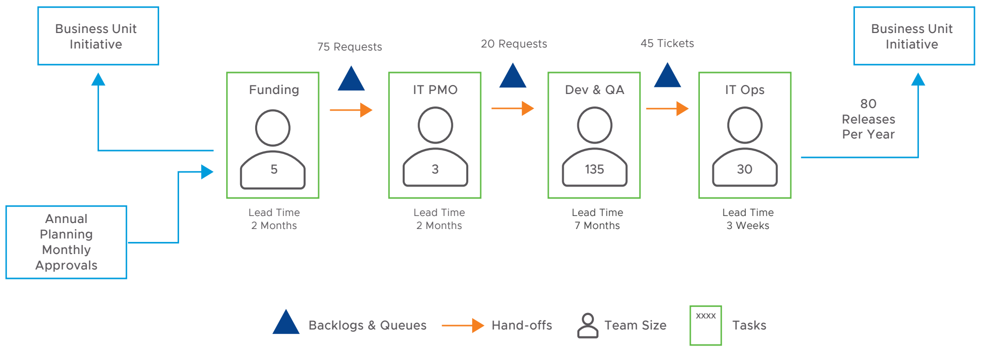


FIGURE 5: A value stream for traditional custom app development.

Initial set of objectives and key results

Clear and quantifiably measured goals help the team stay focused on tangible outcomes, not busy work. As such, these goals are critical to achieving realizing iterative results.

Our clients have seen success with the objectives and key results (OKR) format. We start by assembling a cross section of the team (stakeholders, infrastructure, development, business) into a workshop. The first task: Map near-term goals (mission and high-level objectives) along with specific, quantifiable key results. From there, the sub-team prioritizes what can (and should) be accomplished within the first weeks of the initiative. Be as specific as possible. Ensure your OKRs speak to results and avoid (as best you can) using yes/no metrics that state if something is done or not.

How to replatform your applications

Make the minimal testable changes to each application required to run it on cloud

The first phase of the app modernization efforts moves the initial sample set of applications, those candidate apps selected with the process previously described. Migrated apps should work identically, or better, on cloud than on their current runtime. You can achieve this result even when your apps only conform with a few cloud native factors. In parallel, we work to deploy the required changes with an automated CI pipeline.

Establish three work tracks and apply their respective OKRs

There are three parallel work tracks, each with their own OKRs:

1. Technical **–** The application modernization team will work in week-long sprints. A first app can generally be rehosted or replatformed within hours or days. OKRs for this work track typically measure the number of apps that have been moved.
2. Process – Work through process and policy issues required to get the migrated apps into production. This might include release management; infrastructure issues (network, firewall, DNS); security considerations (OSS, credential management, code scans); application telemetry (logging, health monitoring); risk mitigation (standards adherence, regulation); and business issues (business validation, training readiness). Analysis might result in a new CI pipeline, the adoption of advanced deployment techniques (blue-green, canary) or automated dependency management. OKRs might include pipeline improvements to increase delivery speed or reduce time to resolution.
3. Patterns – Popular technologies such as Java and .NET use standard architectures and employ common messaging patterns. As a result, the technical challenges you solve (e.g., how to modify JBoss code to run on a chosen cloud technology) can be documented as a set of patterns and be reused broadly. This action helps accelerate and de-risk future efforts. This is why you want your initial apps to represent most archetypes in the portfolio. Once the apps have been migrated, the resulting patterns can be used to transform thousands more apps. OKRs should cover documentation and usage of these patterns.

Automate testing to boost code quality

There may be little to no automated test coverage for an existing application portfolio. Retrofitting full coverage is unrealistic. Instead, when writing new code, you should include unit and integration tests, preferably using *TDD practices*. All migrated code should include smoke and health check tests for backing services and, if possible, acceptance tests.

Integrate testing into the CI pipeline wherever possible. This way, testing becomes a standard part of release management. Push migrated applications to production. Start to transition the central QA role to a more exploratory, functional testing role. Document your new testing practices. This will help you standardize modern QA practices throughout your organization as the app transformation efforts scale up.

Improve your continuous integration automation

Many large organizations have release processes that take weeks or months. These processes often become more complex and more opaque over time. Revisit, challenge and make updates to your path to production process model for the migrated apps. This way, you can continuously eliminate wasted effort through automation. Add incremental efficiencies by first automating human-centric, manual processes. We've seen cases where an eight-month release process was reduced to a few weeks, and further shortened to days.

A full tool chain must be available to push to production. This could be an existing tool chain provided by the platform team, a new tool chain selected by the app modernization team or some combination of the two.

Refactor and optimize complex systems

The goal of modernization: Make testable changes to applications and make them run natively on cloud, aiming for 15-factor compliance. This effort is commonly summarized as break monoliths down into microservices.

Refactoring typically results in decomposition of an existing system wherein business logic (and data) is refactored into domain-specific services. Typically, these are long-lived core systems. They tend to be important to the business. The status quo (costly to operate, difficult to improve) is unacceptable. Further, the systems can suffer from lackluster uptime and are near-impossible to scale. These monoliths have served the business well over the last decade, but you need a new model for the next 10 years. That's where microservices come in. How can you navigate this transition cleanly? Simple: by gradually replacing your monolith with microservices.

This process goes something like this. Each resulting microservice often gets its own database(s) to store information it cares about. Collections of related microservices that live in multiple bounded contexts can use eventing to keep data synchronized across data stores. New microservices and replatformed slices will continue to work seamlessly with legacy code until the entire monolith has been moved.

Swift methodology yields a shared understanding of desired system behavior

Swift is a set of lightweight techniques, using agile and *Domain-Driven Design* (DDD) principles to help teams plan enough to get started, and organized around a backlog of work. These include:

1. *Event storm* the system, using language that business and technical people understand.
2. Conduct a Boris exercise that models the relationships between capabilities in a system.
3. Conduct a SNAP that documents the technical capabilities identified during Boris in real time.
4. Identify tactical patterns, like anti-corruption layers and service choreography.
5. Define *OKRs*.
6. Create a backlog of prioritized user stories tied back to OKRs.
7. Start hands-on experimentation, feedback and iterative progress.
8. Swift aligns business leaders and technical practitioners. Use this approach and, at its conclusion, you'll have an architectural plan that maps future goals with the way the system wants to behave. We've found this to be especially important for critical systems modernization.

Information gained through the use of Swift informs decisions on how to organize development teams and prioritize stories (from both a business and technical perspective). It's also helpful as a catch-all way to define a path between the status quo and the desired state.

Storming uncovers what matters: Critical input for any modernization project

Event storming is a cross-functional facilitation technique. It can be applied to a business, a process or a system. In the context of modernization, we use event storming to help reveal logical entities, bounded contexts, trouble spots, questions and starting points. But in simple terms, event storming helps you make sense out of a large mess (like a process or system, for example) and get consensus on what's important as you begin to find the scope of work.

A facilitator leads a mixed business and technical audience through a conference room exercise, documenting the logical flow of a system (or process) from end to end. At Tanzu Labs, we worry less about strict adherence to event storming jargon and rules. We use the exercise to quickly make sense of complexity. Getting people together in person is hard; expert facilitation assures that the best possible result.

Once the system scope and problem context is captured, we proceed to identify notional service candidates. The service candidates will be instrumental to the Boris exercise. This effort is discussed next.

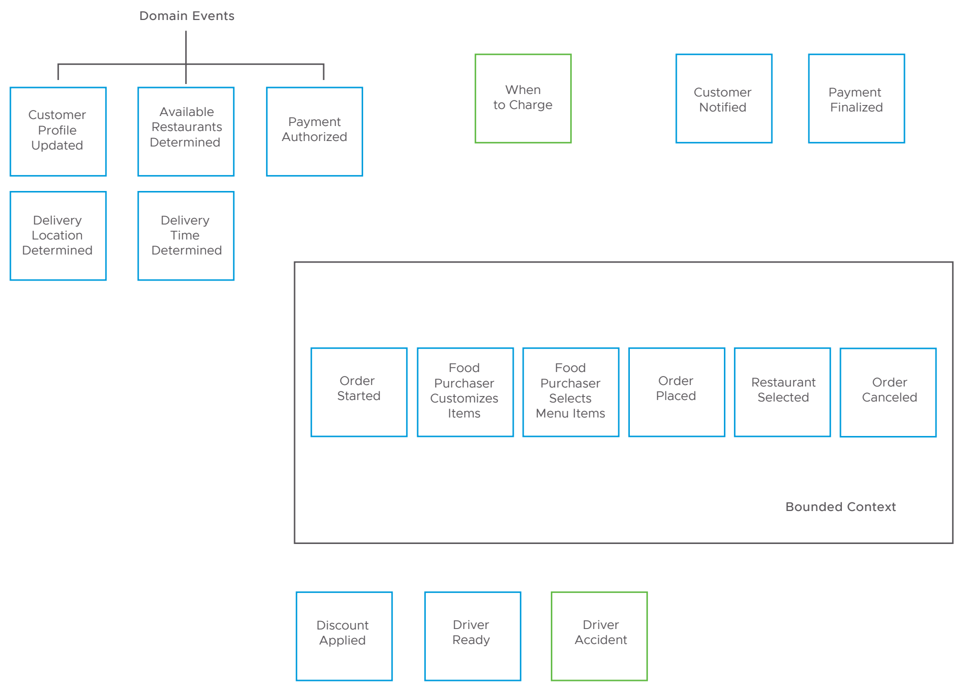


FIGURE 6: A simple event storming example.

Boris exercise models the relationships between system capabilities

The Boris exercise (name inspired by *this song*) helps us identify system components and relationships. A Boris exercise uses graph theory to model the relationships between the capabilities in a system. It generates information about how the system wants to be designed and attempts to avoid the tendency of designing a solution before really understanding the problem. Similar to event storming, Boris depends on live collaboration, a lot of sticky notes and working space (usually a conference room).

The Boris exercise uses insight discovered by event storming and graph theory to identify system components and model their relationships. Colored tape connects sticky notes to indicate communication paths (e.g., direction) and types (e.g., asynchronous). This yields a system diagram that often resembles a spider's web. As the diagram is built, the team performs SNAP analysis (discussed next) to rapidly document findings.

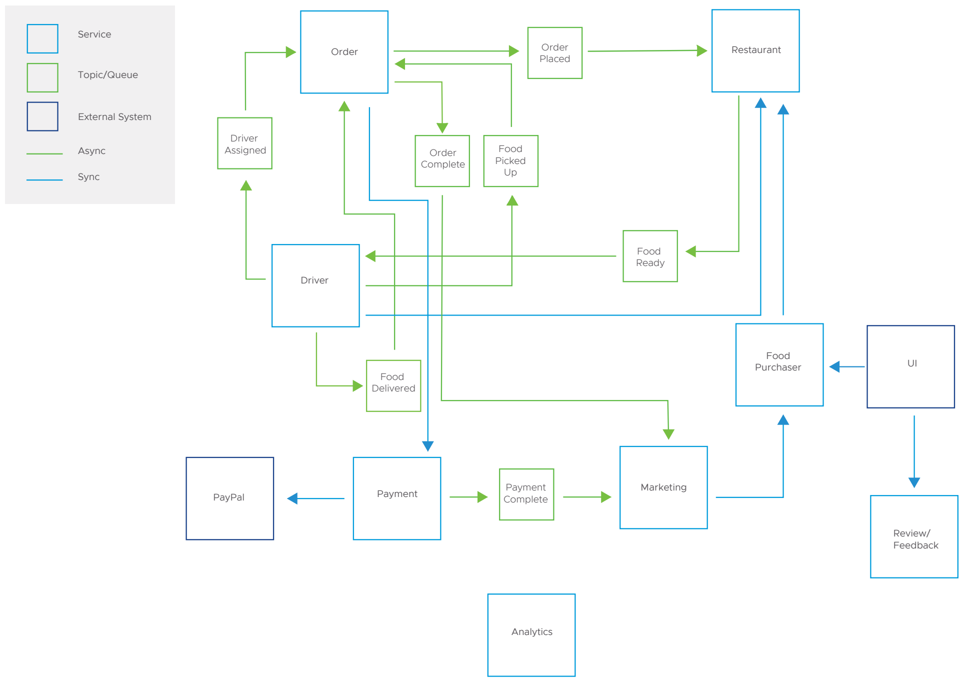


FIGURE 7: Sample Boris diagram based on event storm output from Figure 6.

SNAP documents the outcomes of a Boris exercise

Remember SNAP from our discussion of replatforming? It plays a role in modernization as well. SNAP is used to quickly document the outcomes of a Boris exercise in real time. Information is often grouped into APIs, data, pub/sub, external systems/UI, stories and risks. The key artifact is a poster-sized sticky paper on a conference room wall, with one SNAP per node or service depicted on Boris.

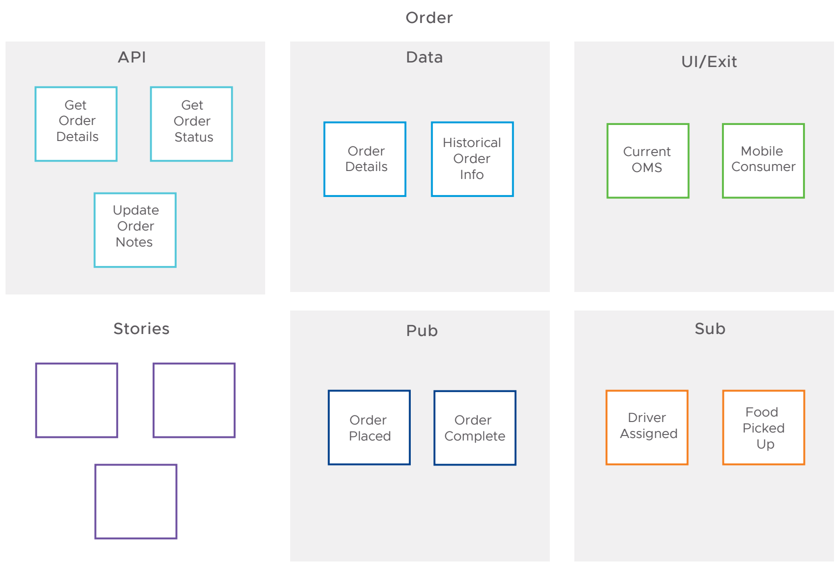


FIGURE 8: Sample SNAP output.

Thin slices of modernization, shaped by Swift and captured as MVPs

Thin slices are short domain event flows. Think of them as the architectural components required to produce those events. Thin slices are informed by event storming, Boris and SNAP activities. They become actionable when captured in the backlog as MVPs or collections of stories. We'll partner with your team to identify and prioritize the thin slices, with an eye to balancing business value, technical risk and effort. The goal is to incrementally move the system toward behaving the way it wants to; the implementation of each successive slice gets us that much closer to this goal. As we define the slices, we also discuss tactical implementation patterns (e.g., anti-corruption layers, Facade, Proxy, Strangler, etc.), risks and challenges.

Initial backlog gets your team hands-on in days

Next, you'll translate many sticky notes into an initial backlog that's prioritized to guide the implementation of thin slices. The backlog contains user stories, component stories for implementing architectures, and experiments (spikes) to address challenges and risks. It's also just large enough to get hands-on work started. Your team will then proceed to quickly deliver functionality, and subsequently prove or disprove assumptions made about the notional architecture. Your backlog is a living artifact; it is continually groomed, and additional stories are added and prioritized to incrementally advance thin slice implementation.

Notice a pattern? Backlog grooming, event storming, Boris, SNAP and slice definition are iterative processes. You'll repeat them as many times as necessary during a system modernization. These methods are most effective when conducted by cross-functional teams that have strong executive support. The practices produce working software, flexible architecture, and a catalog of recipes and patterns.

Use quantitative measurements to track your success

We recommend defining (and continuously refining) OKRs for each step. The ideal OKR is a quantitative measurement that covers process, time and cost improvements. OKRs should provide fine-grained insight, for intended project outcomes, that roll into the broader mission.

How do you know what good looks like? We like the metrics offered by the *DevOps Research and Assessment (DORA)*. Here are a few signals that show your effort is succeeding:

- Increased deployment frequency – More software releases this quarter than last quarter
- Release management efficiency – Lower lead and process time, fewer steps and hand-offs
- Improved operational metrics for transitioned apps – Mean time to recovery (MTTR), mean time between failures (MTBF), support upgrades and so on
- Improved security – Faster patching, zero downtime upgrades and so on
- Efficient infrastructure usage – Higher density compute, auto-scaling and cost reduction

VMware Tanzu Labs helps organizations all over the world to modernize large systems across entire portfolios. However, we do not have all the answers; we learn as much from our customers as we teach them. Customer journeys such as *Liberty Mutual* have informed much of the advice discussed in this white paper.

Next steps and recommended reading

Take a look through the following to learn more about how we partner with organizations like yours:

- *Application Modernization at VMware Tanzu* [Webpage]
- *How AirFrance KLM is Modernizing 2,000 Legacy Applications* [Webinar]
- *App Modernization with .NET Core: A Journey from Mainframe to Microservices* [Webinar]
- *DICK'S Sporting Goods: What is the Future of Retail in a Cloud App World?* [Webinar]
- *Monolithic Transformation: Using DevOps, Agile, and Cloud Platforms to Execute a Digital Transformation Strategy* [eBook]
- *Microservices eBook: Migrating to Cloud-Native Application Architectures* [eBook]
- *The Top Ten App Transformations* [Blog]
- *How a Large Fintech is Breaking Up Monolithic Applications into Microservices* [Webinar]
- *Application Modernization Recipes* [Technical Deep-Dive]
- *Application Modernization Should Be Business-Centric, Continuous and Multiplatform* [Gartner Report]
- *How The Home Depot Became a Digital Powerhouse* [Forrester Research Report]
- *Tanzu Application Analyzer: Your Forensic Source Code Analysis* [Blog]
- *VMware Tanzu Solutions: Why You Should Treat Platform as a Product* [White paper]

If you have insights about application modernization you would like to share, please contact us at *tanzu@vmware.com*.