Resilient and Operationally-Scalable Orchestration

Key Objectives - Can't Live Without:

Self-service remote troubleshooting and failure provisioning

Ability to re-start from the failure point or abandon with full, automatic cleanup

Built-in Upgrade and baseline (OS/Middleware) management

Built-in application configuration (VM/BM), clone/fix (VM image based) or Containers (Docker)

Problems:

Large scale operations with full automation creates severe constraints on underlying systems and services which cause a high number of failures during application provisioning and configuration. The changes could include a password or a key change in one of the shared services, an upgrade of the service with a minimal downtime or even a script error, among many others.

All of these create a set of issues with automation usability, leading to high volume of manual clean-up and recovery operations, costly re-provisioning and, eventually, losing trust in automation. Although it's important for production operations, it can also create major issues with CI/CD where building pipelines could exhibit intermittent failures and become heavy burden on the product development team.

Solution:

It is absolutely essential for the Application Management solution to support **resilient provisioning**. This includes ability to fail gracefully while notifying a user who should be given an opportunity to fix the issue (for example, update the script or a parameter, change password, etc.). The user should be able to continue provisioning from the failure point, going to the next step, once apply the fix, or from a custom step in the flow.

Alternatively, user should be able to abandon the operation, while system automatically releasing all the resources that have been used during provisioning, all without human intervention. User should be able to manually intervene and to either block a complete clean up or to be allowed manual clean-up of failing resources. The system should always have its own state of "clean-up" making sure no "orphans" have been left in the database.

Multi-tenancy with Guarantees

Key Objectives - Make Sure You Have:

Self-service for BUs with AD integration

Shared services for Infrastructure Operations

Data model adopted for “dual” ownership and separation of concerns between BU and Infrastructure Operation

Problems:

In a fully automatic operation, it is absolutely essential to establish trust between Infrastructure Operators and Application Operators. Conflicts typically come from sharing responsibility for resources between Infrastructure and Application Operation. Other conflicts can stem from lack of resource isolation, “noisy” neighbors as well as continued management conflicts.

Solution:

Technology that allows creation of real enterprise tenancy model where Infrastructure Operation is exposing high level abstractions to the tenants for resource management and is adding policies and guardrails for drawing resources. The Infrastructure Operation need to have a view into tenant operations to ensure that limited impact on management changes.

For example, Infrastructure Operation may designate a “shared pool of storage resources” to a tenant, but set up policy for usage of the pool by a given tenant. Thus, isolation would be enforced automatically, without a human in the loop. At the same time, it would assure tenant’s needs for secure guarantees in its operation without impact from other tenants. Using policy-based approach would also ensure agility and speed of application delivery.

Finally, it is essential for a tenant to control security policies for its own applications, while managing these policies as well as isolation between different tenants. For example, a set of shared credentials can be managed by Application Operation for a group of applications and, at the same time, can be different from the credentials using by another tenant. Therefore, VM access or access to any services on the stack (including shared services) would be able to provide tenant with unique and selected credentials.

Patterns/Blueprints as a Product

Key Objectives - Utilize Efficiency Of:

Composition/Components, Platforms and Applications

Full lifecycle management (with version control) – for Dev, Test, Prod and Upgrades

Cloud Independent Blueprints

Multiple Deployment Profiles

Problems:

Traditionally, patterns and blueprints have been interpreted as a set of textual instructions and code that have been created and managed in a source control system. It is a reasonable assumption, but high complexity of such models is not really providing users with repetiveness and the ease of use. Such blueprints have usually deteriorated over time and left users with high cost and low quality code bases. In a way, this would be identical to using "assembly" language to program the computers. It looks powerful and achievable on the surface, but the artifacts would be hard to maintain, reuse, re-purpose and extend. Furthermore, this model is fragile and requires deep expertise to maintain and manage.

Solution:

A way more efficient model is to treat Blueprint and Patterns as objects of system management that is designed to simplify its usability and maintenance. Since Blueprints represent complex interconnection between application components, infrastructure, policies and other applications aspects, managing it is not a trivial task. A few key points to mention:

First, we want to ensure that a single Blueprint is really… a composition of other blueprints. Specifically, we need to model application components as separate blueprints to ensure that they can support standard protocols for installation, upgrade, scaling, and deployment of application artifacts. Components is also providing support for HA, connectivity and security. Typically, components represent highly reusable middleware like Tomcat or MySQL.

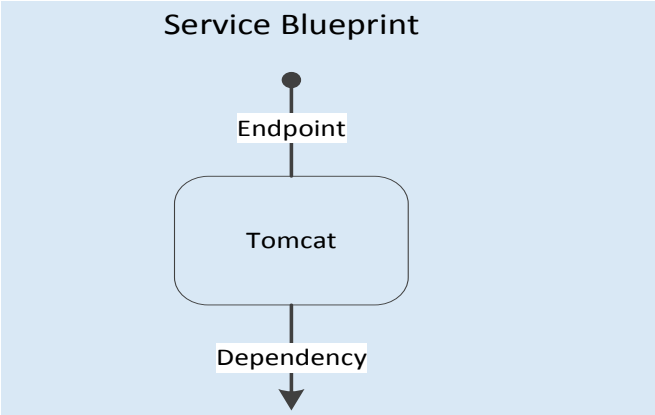
Next, we want to have ability to compose the Platform blueprints. The platform would span deployment groups and handle all aspects of application management and configuration. The goal is to make this process as trivial as it would be useful.

Finally, we need a way to create an application, from the platform, with focus on application specific artifacts. This approach ensures encapsulation of responsibility, high level of reuse and ease of blueprints consumption.

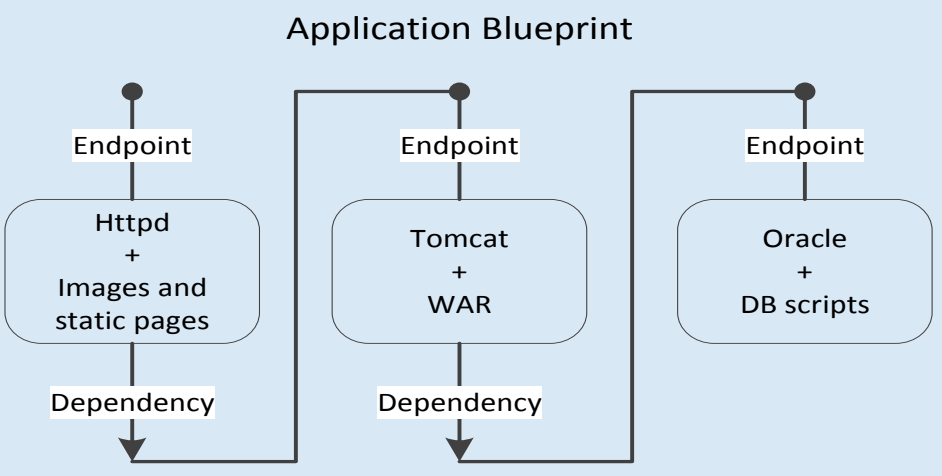
UNITY KEY BENEFITS

Since blueprints become a complex of objects collection, their lifecycle management could become a formidable task to perform. It is essential to enforce version control, while delivering easy to use mechanism for version advancing and release of components and compositions. The ultimate goal though is to create a simple mechanism to manage blueprint lifecycle without forcing this tedious job on developers and third party tools. This also includes testing and troubleshooting, refactoring and blueprints upgrading.

The expectation is that Blueprints would become reusable across Clouds and Use Patterns and would easily support compact and single VM deployment. It would also be able to support more elaborate PROD deployment without extra point of management.



Service Blueprint is a single service provisioning definition, contains all media, scripts and parameter definitions necessary for service deployment



Application Blueprint defines specific customizations applied to reusable Platform Blueprint i.e. WAR file deployed into Tomcat

High Level Abstractions

Key Objectives - Model On

Infrastructure Resource Management (Providers, Resource Pools, Resource Policy, Tenancy)

Application Management (Component/Platform/App, Deployable Artifacts, Environment Types)

Security as a Resource (Security profiles, Isolation zones, Endpoint security, Service Accounts)

Problems:

Application Management is an interconnect of many complex systems. Since most of the components have APIs, one can assume that all we need is a glorified workflow engine with a plug-in model and easy coding. While it is possible to use this model to automate a few flows with some fixed parameters, the model breaks apart very quickly and become completely non-scalable in case of large operations.

Solution:

Instead, we are proposing the Application Management framework and technology to simplify and to operate it without any coding

The modeling starts on the Infrastructure level. Since provisioning draws resources for Tenant's application from a common pool, we provide representation of these resource pools in the system. Given that resources may have many aspects and variants across clouds, it would be impossible to simply expose all of their aspects and permutations. Instead, a high level abstraction is created and defines its policy and guardrails. The abstraction can be easily handed over to tenant for further use. Its representation is easily understood by the tenant and become an ultimate solution for deployment, while allowing quick and easy policy-based automation and configuration.

The abstraction can be managed at the Application level, for example, performing separation of Application Platform from Application Artifacts. The latter is used to isolate application specifics, while the former is to facilitate the reuse and encapsulation of application configuration and deployment complexity. Such abstractions dramatically simplify the usage, makes pattern developer aware of the complexity and works extremely well to encapsulate it in the Platform. It allows developer to decide on a particular HA deployment model or security constraints on communication between two components. All the complexities would be hidden from the general developer community given requirement to produce only the set of application specific artifacts and to use the standard build process to deploy them.

Finally, it is absolutely necessary to treat Security as Resources. For example, instead of handing over credential management to individuals, the system would allow to create appropriate resource configurations for each tenant, or for each group of applications, or even for individual applications. The credentials would be generated and consumed by the management system without human interaction, using policies and enforcing rules.

Key Principals

From	To
Diagrams and specs	Visual Composition
Snowflakes with exponential complexity	Heavy reuse, pattern based
Explicit Infrastructure Management	Invisible Infrastructure
DevOps coding for every app	Very little to none application specific DevOps code
Choose between image cloning systems or dynamic node configuration systems (Chef)	Images are automatically generated as a cache but anything can be built from scratch. Docker included.
Complexity of change, upgrade and patch	Built-in upgrade and patch model with testing
Disconnected tooling	Integrated with CI/CD, dev., QE and AppOps

Framework for extending Data, Services and UI/CLI

Key Objectives - Focus On:

Ability to add Custom Data and Services with REST API (Example: Custom Asset Management objects)

Ability to add Custom UIs and CLI operations (e.g. Storage Management UI)

Ability to extend Providers (including Data, Services and UI), Policies and other system components

Solid technology stack with built-in HA and DR

Problems:

The volume of tools, consoles and dashboards have been multiplying each week. This creates adverse effects for communities that participate in the Application Management network. There is a strong need to find a way to reduce complexity as new solutions keep rolling out. In effort to simplify user experience, attempts have been made to consolidate (and integrate) the related experiences along two axes: Tenant and Infrastructure Operations.

Solution:

In this case, one should look for a technology that provide a mechanism for extensions of both experience and custom use cases. For example, Storage Management operations requires extensions for storing Data (e.g. Device, Vendor, Provider), adding Services with REST API (e.g. add/discover Device) and UI for effectively managing devices across vendors. Ideally, Application Management technology provides “plugability” for such custom operations all the way through API and UI. However, contrary to the idea that we may want to create a monolith, we expect the integrated experience to be achievable by federating such capabilities.

Another example is Traceability which is nearly impossible to think that it can be achieved in the standard UI, right out of the box. Since the state and relations are distributed across multiple components, one needs to create a data warehouse that would be able to support visualization and analytics for this type of data. And we still want to be able to add a view in the Tenant experience that seamlessly integrates such visualization (Application-to-Infra) from the end user perspective. For the Infrastructure operators, this experience should also be integrated, although visualization would also support Infra-to-Application traceability.

Finally, we should be able to use the framework to extend technology itself in a well-defined way. For example, implementing custom scheduling would be done using same coding and run-time model as technology itself.

Ideally, we see application platform’s stack that’s used by technology with all the built-in capabilities for data management, service APIs, security, HA, logging, while tracing its immediate availability as a separate consumable platform for other extensions. In this case, the user can receive substantial benefits from the solution, while getting simpler, cheaper and high quality operation framework.