# Automated Virtual Server Provisioning Solution – Architecture Design Document

**Prepared for:** Internal Architecture & Automation Review  
**Prepared by:** *(Your Name)*, Infrastructure Architect  
**Date:** August 2025

# Preface

This document presents proposed architecture and workflow for automating virtual server provisioning using ServiceNow as the central orchestration platform. It is written from the perspective of the **build process**, focusing on how requests are captured, validated, approved, and coordinated across existing automation systems to deliver a consistent, repeatable outcome.

## The design assumes:

* Existing backend automation already exists for provisioning, network configuration, IP address management, and related tasks (e.g., automation platform, configuration management tool, IP management system, AIX build processes).
* These backend capabilities are owned, operated, and maintained by other teams.
* ServiceNow will serve as the single point of entry and control for the process, ensuring that all work is initiated, tracked, and auditable from one location.
* Integrations between ServiceNow and backend systems will leverage established methods already supported by other teams.

The content takes a **technology-agnostic approach** to backend execution. Wherever possible, it describes *what needs to be done* rather than prescribing *how it must be implemented*. This avoids dictating tooling or technical pathways while still providing enough functional clarity for all stakeholders. The tone and structure intentionally acknowledge other teams’ ownership of their systems, recognizing their existing work and ensuring that the orchestration design complements, rather than replaces, current processes.

This architecture is intended as a collaborative foundation for discussion, refinement, and alignment across all contributing teams. It is flexible and extensible, allowing for integration with other enterprise platforms or evolving automation capabilities, while remaining grounded in shared governance and operational best practices.

# Executive Summary

The Virtual Server and AIX Provisioning Automation project will transform the current manual server build process into an end-to-end automated workflow using ServiceNow as the orchestrator. This design introduces a self-service provisioning model for both VMware virtual machines and IBM AIX servers by integrating ServiceNow with the existing backend automation for virtual server deployments and IP address management. The goal is to significantly reduce fulfillment time (from weeks to days or hours) and eliminate manual errors by orchestrating requests, multi-team approvals, IP allocation, VM/AIX builds, and configuration updates in a unified platform. Key outcomes include faster delivery of infrastructure to projects, improved consistency and compliance in builds, and enhanced transparency for stakeholders throughout the process. This document provides a high-level overview for executives as well as detailed technical design for implementation.

## Objectives and Strategic Rationale

The solution’s core objectives and the strategic rationale for this initiative are outlined below:

* **Accelerate Delivery:** Dramatically shorten server provisioning timelines from the current 2–4 weeks to days or hours. Faster delivery of servers will unblock dependent initiatives and speed up project timelines, directly supporting business agility and time-to-market goals.
* **Improve Operational Efficiency:** Minimize manual effort by automating intake, approvals, and build tasks. By cutting 40–60% of the labor currently spent on coordination and builds, skilled engineers can be redeployed to higher-value projects, yielding cost savings and productivity gains.
* **Enforce Consistency and Compliance:** Ensure every build adheres to naming conventions, security policies, and configuration standards. The automated workflow embeds governance (approvals, data classification checks, etc.) so that each deployment meets internal standards and regulatory requirements without ad-hoc checks.
* **Enhance Agility and Self-Service:** Empower teams with a one-stop, self-service catalog for standardized server requests. Transparent approval tracking and on-demand fulfillment improve user experience and operational velocity, aligning with our digital transformation and Infrastructure-as-Code strategies.
* **Risk Reduction:** Reduce errors and rework by removing error-prone manual data entry and handoffs. Standardized automation results in more reliable, repeatable builds, decreasing the risk of misconfigurations and subsequent incidents. All actions and approvals are logged for audit, providing a clear compliance trail.
* **Strategic Alignment & Future Growth:** Laying this foundation now provides a platform for future enhancements (e.g., hybrid cloud integration, broader Infrastructure-as-Code adoption). Early implementation maximizes ROI by immediately capturing time savings on every request and prevents accumulation of technical debt from continued manual processes. The initiative positions the organization to meet FY25 operational excellence targets and scales for growing demand.

In summary, investing in this automated provisioning solution now will modernize our infrastructure delivery, improve service quality, and free up engineering capacity – a strategic move that enables both immediate benefits and long-term agility.

## Overview of Current Challenges

Our existing server request and deployment process is **slow, siloed, and error-prone**, underscoring the need for this overhaul. Key challenges in the current state include:

* **Lengthy Provisioning Lead Times:** End-to-end fulfillment typically takes 2–4 weeks. Requests may start in ServiceNow but then languish through manual design reviews, email approvals, IP/DNS setup, and automation platform provisioning. Each hand-off (waiting for architectural review, network IP assignment, etc.) adds delays, resulting in frustrated end-users and project bottlenecks.
* **Fragmented Data Gathering:** There is no single source of truth for request details. Teams rely on Excel design forms and email threads to gather specifications. For example, AIX builds require separate design documents and meetings, while VMware VM specs (IP, VLAN, storage, etc.) are manually compiled by engineers. This scattershot approach leads to repetitive data entry and missing information.
* **Multiple Approval Bottlenecks:** Every request must be reviewed by Architecture, Network, Security, and more, but today this is coordinated via ad-hoc emails and meetings. Sequential approvals often stall waiting on feedback or additional info. A lack of a unified workflow or automation means it’s easy for requests to get stuck or ping-pong between teams for clarifications, significantly extending delivery times.
* **High Risk of Errors and Rework:** Manual transcriptions of data between forms, ServiceNow, automation platform, and IP management system lead to frequent errors (typos in hostnames, wrong VLAN IDs, etc.). These mistakes often surface late in the build process, causing failed deployments or misconfigured servers that require troubleshooting and rebuilding. Such rework consumes additional time and undermines confidence in the process.
* **Siloed AIX Provisioning Process:** IBM AIX server requests follow a completely separate, labor-intensive path. They involve unique spreadsheets, specialized reviews, and manual builds via NIM (Network Installation Manager) or scripts on Power systems, often outside of ServiceNow tracking. This parallel process means poor visibility in the CMDB and inconsistent governance. AIX builds suffer the same delays and errors, compounded by platform-specific complexities (LPAR configurations, VIOS, etc.), yet remain largely unintegrated with the standard request system.

These challenges highlight the inefficiencies and risks of the status quo. The lack of a unified, automated pipeline not only delays projects but also consumes significant expert time on routine tasks and opens the door to misconfigurations and compliance gaps. Addressing these pain points is critical to meet the organization’s goals for agility, reliability, and auditability in infrastructure delivery.

## Target State Architecture and Design Principles

In the target state, server provisioning will be handled by a **unified orchestration architecture** centered on ServiceNow, with best-of-breed tools handling the specialized tasks (VMware automation platform for virtualization and IP management system for network services). All server requests (Windows, Linux, and AIX) enter through a single ServiceNow portal and follow a standardized fulfillment workflow. At a high level, ServiceNow will manage the front-end form, dynamic approvals, and overall process logic, while automation platform and configuration management tool/other tools carry out the actual server builds and configurations in the backend.

**Design Principles:** The solution design adheres to several key principles to ensure it meets current needs and future scalability:

* **Single System of Engagement:** ServiceNow serves as the **single entry point and coordinator** for the entire process. This centralizes all request data, approvals, and status tracking in one place, improving visibility and eliminating the previous patchwork of emails and spreadsheets.
* **End-to-End Automation:** The architecture automates every feasible step from request submission to deployment. Manual intervention is only required for truly exceptional cases or legacy processes. This ensures a **consistent, repeatable workflow** that reduces errors and accelerates delivery.
* **Policy-Driven Orchestration:** Built-in **governance and compliance** checks are applied at each stage (e.g., enforce data classification rules, require security approval for high-risk builds, auto-generate compliant hostnames). This guarantees that corporate standards are met **by design**, rather than relying on after-the-fact reviews.
* **Modular Integration:** The system is composed of loosely coupled components ~~(ServiceNow, automation platform, IP management system, configuration management tool)~~ connected via APIs and a **~~Mid~~ Server** for secure communications. Each integration (IPAM, virtualization, etc.) is encapsulated, allowing components to be swapped or updated independently (for example, if we adopt a new IPAM tool or upgrade ~~automation platform~~, the overall workflow remains intact).
* **Extensibility and Platform Coverage:** While initial automation targets VMware VMs, the design accommodates **AIX and physical servers** through conditional logic and task routing. As AIX automation capabilities (e.g., via IBM PowerVC or configuration management tool modules) mature, those can be incorporated into the same framework. The architecture is flexible to onboard additional platforms or cloud environments in the future, ensuring longevity of the solution.
* **Security by Design:** The architecture strictly follows security best practices (especially for DMZ access – see below) and **least privilege** principles. Credentials for integrations are stored securely, network access is tightly controlled, and all actions are auditable. No automation components reside in the DMZ; instead, secure proxies (s and API contracts) are used to reach out as needed. This minimizes the attack surface while enabling controlled automation across security zones.

By adhering to these design principles, the target architecture will significantly reduce provisioning time and effort while maintaining robust control and compliance. The diagram below provides a high-level overview of the major system components and their interactions in the target state.

**Target State Overview:** ServiceNow orchestrates the end-to-end provisioning process, interacting with IP management system for IP/DNS management and VMware automation platform for VM deployment, and updating the CMDB for asset tracking and operations.

*Figure: High-level integration architecture – ServiceNow (central orchestrator) interacts with IP management system for IP address management and DNS, triggers VMware automation platform for VM provisioning on vCenter, and updates the ServiceNow CMDB. A ServiceNow provides secure connectivity to on-premises systems.*

## System Components

This section describes the key system components and integrations in the solution, detailing their roles and how they work together in the automated workflow:

### ServiceNow Service Catalog & Approval Workflows

**Role:** ServiceNow provides the user-facing **Service Catalog** for request intake and the **Flow / Workflow Engine** to orchestrate the process.

**Capabilities:** A new catalog item (e.g., “**Request New Server**”) will capture all required information in a structured form, replacing the current Excel design documents. The form includes fields for business details (project info, justification, etc.), technical specs (OS, CPU, RAM, storage breakdown, environment, network tier), and any special requirements (e.g., High Availability, clustering for AIX). Many fields use dynamic choice lists or dependencies – for example, selecting “AIX” as OS might prompt for an LPAR size or identifying an existing frame, and choosing “DMZ” network tier may flag additional security reviews. The form also auto-generates certain values: a **hostname** following our 16-character naming convention is suggested based on inputs (site, application acronym, env, sequence), which the user can review.

On submission, ServiceNow will initiate an **approval workflow** tailored to the request. The platform’s Flow Designer (or Workflow Editor) will programmatically route the request through multi-stage approvals. **Embedded approvals** include Architecture (for all requests), then conditional approvals for other teams depending on the request data: e.g. Network team approval if a subnet or VLAN is not specified or if the server will reside in a sensitive network (like DMZ), Security approval if the data classification is High or it’s a production server, Storage approval if large or non-standard storage is requested, etc. The workflow supports parallel approvals wherever possible to speed up the process (for instance, Network and Security reviews can happen concurrently if there are no dependencies). Each approver receives a task in ServiceNow (and notification email) and can approve or reject with comments. SLA policies are attached to each approval step (e.g., Architecture team to approve within 2 business days) with automated reminders or escalation if an approval is delayed. By managing approvals centrally in ServiceNow, we eliminate the previous email chase and ensure **no server is built without the required sign-offs**.

If any approver rejects the request, the workflow stops and the requester is notified with the reason, enabling them to correct and resubmit if needed. Approved requests move automatically into the provisioning phase.

### Secure Automation Access to DMZ (FRB DMZ Policy Alignment)

**Role:** This component refers to the **network and security configuration** that allows our automation tools (ServiceNow , automation platform, configuration management tool) to safely reach servers in the **DMZ network segment** during provisioning, in compliance with Federal Reserve Board (FRB) security policies.

**Approach:** The FRB’s DMZ policy mandates a **highly restrictive, controlled access** model (“deny by default”). In the target design, no orchestration or management system is deployed directly in the DMZ. Instead, all automation originates from the trusted internal network and reaches into the DMZ through tightly governed channels. Specifically, within our Cisco ACI-based data center networking, we will establish an **ACI Contract** between the “Automation Systems” endpoint group (which includes the s, automation platform appliances, and configuration management tool control nodes in the internal zone) and the “DMZ Servers” endpoint group[[4]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=2,ACI)[[5]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=3,Policy). This contract explicitly allows only the minimum required protocols from automation to the DMZ: **SSH (TCP 22)** for Unix/Linux management, **WinRM over HTTPS (TCP 5986)** for Windows management, and **HTTPS (TCP 443)** for any API calls or agent communication[[5]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=3,Policy). All other traffic is denied, and absolutely no inbound connections from the DMZ to internal are permitted[[6]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=with%20FRB%20DMZ%20policy%20prohibiting,generic%2C%20broad%2C%20or%20open%20access).

The contract is scoped to the specific source IP addresses of our automation servers – broad subnets or wildcards will not be used[[7]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=Create%20explicit%20ACI%20contract%3A%20Allow,Automation%20EPG%20to%20DMZ%20EPG). This ensures that only our known automation tools can access DMZ hosts, and only on the approved ports. Furthermore, these connectivity rules are complemented by firewall policies to log and inspect the allowed flows, and all access will be auditable per FRB standards[[8]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=5).

On the credential side, **key-based or certificate-based authentication** will be used for any automation tasks on DMZ servers (no passwords over these channels)[[9]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=Key%2Fcertificate,not%20allowed). For example, if configuration management tool needs to configure a Linux DMZ server, it will use SSH keys; Windows WinRM access will use domain certificates or tokens, etc. The automation service accounts are set up with **least privilege**, and are distinct from any interactive admin accounts[[9]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=Key%2Fcertificate,not%20allowed). All secrets are stored in secure vaults (e.g. configuration management tool Vault or automation platform’s credential store) and rotated regularly[[10]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=allow%20broad%20or%20wildcard%20targeting). We will maintain **separation of duties** – any changes to these DMZ access configurations (like modifying the ACI contract) require approval from Security Architecture and go through change management to ensure compliance[[11]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=No%20inbound%20flows%20from%20DMZ,are%20allowed%20under%20any%20circumstance)[[12]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=No%20inbound%20DMZ).

In summary, this component of the design enables the automation to **seamlessly deploy to DMZ-located VMs** (e.g., web servers in a DMZ network) without violating security policies. By using controlled network contracts and hardened credentials, we achieve the necessary connectivity for provisioning and configuration while **upholding a zero-trust posture** in line with FRB DMZ requirements. (See **Section 9: Security and Access Controls** for additional security measures.)

### AIX Support and configuration management tool-Based Automation

**Role:** This solution is designed to handle **IBM AIX server requests** alongside standard virtual server (VMware) requests, ensuring AIX provisioning is tracked and partially automated where possible. configuration management tool is introduced as a key automation tool for tasks not covered by automation platform, especially for AIX or custom configuration steps.

Approach: In the unified ServiceNow catalog, users can select AIX as the OS and indicate if the request is for a virtual LPAR on IBM Power systems or a physical server. The intake and approval process for AIX will be the same as for x86 VMs, providing a single consistent interface. Once an AIX request is approved, the workflow will detect that AIX is needed and branch accordingly. Because VMware automation platform primarily manages VMware vSphere environments, it will not directly provision an AIX LPAR. Instead, the fulfillment might use one of two methods: 1. Automated configuration management tool Playbook (if available): We are planning to leverage Red Hat configuration management tool (with the certified IBM Power/AIX modules) to automate parts of the AIX build process. ServiceNow can trigger the existing automation process for AIX builds, whether via automated playbooks or other methods already in use. This playbook could interface with IBM HMC (Hardware Management Console) or PowerVC to spin up a new AIX LPAR from a template or NIM install, configure networking on it, etc. Using configuration management tool’s agentless approach is attractive because IBM provides supported modules for AIX administration, allowing us to treat AIX configuration as code similar to Linux.  
2. Task Assignment for Manual Build: If full automation isn’t yet available for AIX, the workflow will create a task for the AIX engineering team with all the gathered details. The AIX admins would then perform the LPAR creation and OS installation manually or via existing scripts (such as NIM deployments). They would then update the task or ServiceNow request once the AIX server is built.

Either way, the new AIX server’s data will be captured in ServiceNow (with a new CI in the CMDB) just like a VMware VM, ensuring visibility. Over time, our goal is to **increase configuration management tool automation for AIX**. For example, after the initial LPAR is provisioned (manually or via PowerVC), we can use configuration management tool to automatically configure the OS (setting hostname, networking, users, deploying standard middleware, etc.). Future integration of IBM PowerVC or IBM’s cloud management for Power could allow even the LPAR creation to be API-driven. The architecture is ready for this – we would simply add a step to call the appropriate API or tool when available, similar to how we call automation platform for VMware.

Additionally, configuration management tool can be used for **post-provisioning configuration management** across both AIX and Linux servers (ensuring consistency in firewall settings, installing monitoring agents, etc.), which automation platform could invoke via its orchestration or which we could run as a follow-up task from ServiceNow. This hybrid approach ensures that even platforms outside the core VMware environment are **first-class citizens in the request workflow**, with as much automation as possible, and at minimum are properly tracked and governed through ServiceNow. AIX support in this project closes a major visibility gap and brings AIX into alignment with modern automation practices.

*(Note: Physical server requests, if any, would follow a similar model – the workflow could trigger automation or create tasks for data center teams. The focus is to capture* *all server builds* *in one system, even if the backend fulfillment differs.)*

## Technical Workflow

The end-to-end technical workflow is outlined below, breaking down each stage from the initial request to completion. This workflow is implemented in ServiceNow’s flow engine, invoking integrations with external systems at the appropriate points and including error handling throughout.

*Figure: Automated provisioning workflow – this flowchart depicts the sequence of steps and decision points in the fulfillment process, including dynamic approval branches (diamonds) and integration tasks (rectangles) for IP allocation, VM provisioning, CMDB update, and notifications.*

### Request Intake and Validation

**Step 1: Request Submission:** A user (e.g., a developer or project manager) initiates the process by filling out the **“New Server Request”** catalog form in ServiceNow and clicking submit[[14]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=A%5BSubmit%20Request%20%28ServiceNow%20Catalog%29%5D%20,Validate%20Inputs%20%26%20Generate). The form is designed to validate inputs in real-time – required fields must be filled, and certain values are checked (for example, ensuring CPU and RAM selections are within allowed ranges, or that a Project ID provided actually exists).

**Step 2: Automated Hostname Generation:** Upon submission, the workflow immediately generates a **tentative hostname** for the server[[14]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=A%5BSubmit%20Request%20%28ServiceNow%20Catalog%29%5D%20,Validate%20Inputs%20%26%20Generate)[[15]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=ServiceNow,prevent%20incomplete%20requests%20from%20progressing). This uses a predefined naming convention script that combines inputs like site code, application acronym, environment, and a numeric sequence. The name is truncated or adjusted to meet the <=16 character policy. This guarantees that even before any human reviews the request, we have a valid hostname reserved. If the requestor provided a custom service/application acronym, the workflow uses it; if that was left blank, the workflow can either flag it for update or proceed with a generic placeholder and follow up later (depending on policy).

**Step 3: Input Validation:** The system checks all inputs for completeness. If any mandatory information is missing or obviously incorrect, the workflow can auto-reject the submission or create a “Needs Information” task back to the requester[[16]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=)[[15]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=ServiceNow,prevent%20incomplete%20requests%20from%20progressing). For example, if an uncommon option is selected (say a non-standard OS build) without required additional details, the workflow pauses and asks for clarification rather than proceeding with incomplete data. This front-loads data quality and prevents the request from moving forward until it’s ready. Once the request passes validation, it moves to the approval phase.

### Approval Routing Logic

**Step 4: Dynamic Approval Chain:** ServiceNow then kicks off the **approval sequence**. The workflow uses conditional logic to determine which approvals are needed based on the request parameters[[17]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=are%20dynamic%20based%20on%20request,details)[[18]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=Security%20Approval%3A%20For%20certain%20environments,team%20approval%20is%20required). All requests will at minimum go through an **Architecture review** (to ensure the solution fits standards). After that, the workflow evaluates conditions:

* **Network Approval:** If the server is in a network-sensitive zone (e.g., DMZ) or the requester did not specify a VLAN/subnet (meaning network team input is needed), a Network Team approval task is sent[[19]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=standards%20). Otherwise, for a straightforward request in a standard zone with all info provided, this step is skipped.
* **Security Approval:** If the requested server will host **High** classified data or is in Production (or DMZ), a Security Team approval is required[[18]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=Security%20Approval%3A%20For%20certain%20environments,team%20approval%20is%20required). This ensures InfoSec reviews any high-risk deployments. Non-sensitive dev/test requests might bypass this.
* **Storage Approval:** If the storage requested exceeds certain thresholds or requires special SAN configurations, the Storage team must approve[[20]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=Storage%20Approval%3A%20If%20the%20request,team%20approval%20is%20inserted). Otherwise this step is skipped for normal storage asks.
* **Platform (Infrastructure) Approval:** The core platform or virtualization team gives the final technical approval on all requests[[21]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=SAN%20configurations%2C%20a%20Storage%20team,approval%20is%20inserted) – confirming resource availability on clusters, etc. This is generally required for all builds as a last gate before execution.
* **Application Owner Approval:** If an application name/owner was specified, an approval task goes to that **business application owner** to confirm the server is needed and appropriately scoped[[22]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=technical%20approval%29%20). This is a governance step to ensure the business side acknowledges any resource costs or licensing implications. In some cases (like a large project) this may be pre-approved or not needed for dev servers.
* **Financial Approval:** If the request is outside predefined cost boundaries (e.g., an unusually large VM or one that triggers significant licensing costs), a Finance or Asset Management approval is triggered[[23]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=). Standard low-cost requests are typically pre-approved financially via policy, so this would be conditionally added only when necessary.

These approvals are orchestrated **sequentially** in the workflow by default[[24]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=These%20approvals%20occur%20sequentially,offs). However, the workflow can parallelize some of them when appropriate. For example, if Architecture approval is obtained, the Network, Security, and Storage approvals could all be requested in parallel (assuming they’re all required), since they don’t depend on each other – the workflow would wait until all parallel ones are completed. On the other hand, we often require Architecture to sign off first before others even see it (to ensure the design is sound), which the workflow enforces by not launching the subsequent approvals until Architecture approval is recorded. ServiceNow keeps track of who approves/rejects and time-stamps everything for audit. Reminder notifications are automatically sent if an approval is pending too long, and if an approver is out-of-office, delegation rules allow backup approvers to act to prevent delays.

The end result is a fully documented approval trail where every required stakeholder has either approved or the request is halted due to a rejection. Once all necessary approvals are **obtained**, the workflow transitions the request into the fulfillment phase.

### IPAM and DNS Assignment (IP management system Integration)

Step 5: IP Address Allocation: With approvals done, the first fulfillment step is to assign network coordinates. ServiceNow (via the ) ServiceNow requests an IP address from the existing IP management process. The workflow already knows which subnet or network to target (from the request’s site/environment info). Using the IP management system API, it finds the next available IP in that subnet and marks it as allocated by creating a reservation or host record in IP management system. This ensures no other process will hand out the same IP. If the requester provided an IP (and policy allows it), the workflow could skip this step and validate/reserve the provided IP instead.

**Step 6: DNS Record Creation:** Immediately after getting the IP, the workflow creates a **DNS host record** in IP management system for the server’s hostname pointing to that IP[[2]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=Story%203,name). This is effectively the automated version of what the network team used to do manually – now done in seconds. By the time this step completes, the server’s name can be resolved via DNS and the IP is confirmed, all logged under the change record.

These IPAM steps remove the need for separate network change tickets. The integration is done using **ServiceNow IntegrationHub** actions if available (IP management system spoke) or via a custom scripted REST call. In either case, credentials are kept secure, and the handles the communication. If any error occurs here (e.g., IP management system is unreachable or returns an error), the workflow will **gracefully handle it**: possibly by retrying a few times, and if still failing, marking the request for manual intervention by the network team while pausing further automation[[3]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=Integration%20details%3A%20This%20is%20implemented,case%20of%20failure%20after%20reserving)[[27]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=6,It%20will%20then). This prevents provisioning from continuing without a valid IP. Successful IP and DNS assignment are recorded in the request’s work notes.

### VM Deployment via automation platform

Step 7: VM Provisioning: Next, ServiceNow initiates the backend provisioning of the server using the existing automation tooling (for VMware targets). Using the integration method discussed earlier (plugin or REST API), ServiceNow passes the final server specifications to the backend provisioning process: e.g., VM template or image to use (based on OS selection), CPU/RAM specs, the reserved IP address, network selection (VLAN/port group), hostname, etc. This call kicks off a deployment in automation platform. automation platform in turn communicates with vCenter to create a new VM, attach the appropriate network (with the provided IP/MAC), and power it on. Any customization scripts defined in the blueprint (such as cloud-init for Linux or Sysprep for Windows) will run inside the VM to finalize the configuration (set OS hostname, join domain, etc.). automation platform handles these tasks autonomously once triggered.

ServiceNow will monitor the provisioning progress. If using the automation platform plugin, the plugin can update the request ticket when the VM is deployed (success or failure). If using direct API calls, the ServiceNow workflow may **poll** automation platform periodically (e.g., every minute) for a status update[[30]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=necessary%20inputs%2C%20like%20the%20VM%E2%80%99s,user%20knows%20things%20are%20moving)[[31]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=includes%20a%20wait%20loop%2C%20polling,user%20knows%20things%20are%20moving). Provisioning typically takes a few minutes. During this time, the request status in ServiceNow can be shown as “In Progress: Building VM…” so the requester knows it’s underway.

When automation platform finishes, it returns a success (with details like the VM’s unique ID or IP) or a failure. On **success**, the workflow moves to the next steps. On **failure**, the workflow will log the error details (from automation platform) and proceed to error handling (skip to Step 10 below). Common failure causes might be lack of capacity on the cluster, misconfiguration in the blueprint, or integration issues – these are expected to be rare after thorough testing, but we handle them just in case.

### CMDB Updates and Notifications

**Step 8: CMDB Update:** Upon successful VM creation, the system ensures the new server is properly recorded in the **Configuration Management Database (CMDB)**. Since ServiceNow is the system of record, the workflow can directly create or update a **CI (Configuration Item)** entry representing the server[[32]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=ServiceNow%20CMDB%20Updates%3A%20A%20critical,improving%20asset%20visibility%20for)[[33]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=5.%20Post,the%20time%20the%20request%20is). Key attributes like hostname, IP address, OS, VM ID, etc., are populated. We will also establish relationships (e.g., link the CI to the requesting business service/application if provided). If we have the automation platform ITSM plugin, it might automatically synchronize CIs for automation platform deployments (the plugin can pull deployed items into the CMDB), but we won’t rely solely on that – our flow will explicitly update the CMDB to ensure immediacy. Additionally, we plan to run a targeted **ServiceNow Discovery** or **IP scan** on the new server shortly after build[[34]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=brings%20vRA%20VMs%20into%20the,the%20business%20service%20if%20provided). This will retrieve deeper information (software installed, serial numbers, etc.) and verify the CI record’s accuracy. By the time the workflow closes, the CMDB will have a complete record of the server in its “Operational” state. This up-to-date inventory is critical for downstream processes like incident management, patch management, and compliance audits.

**Step 9: Notifications and Handoff:** Finally, the workflow sends out **notifications** to relevant parties that the server is ready[[35]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=Notification%3A%20The%20system%20sends%20an,could%20be%20a%20triggered%20task). The requester (and optionally the project manager and application owner) receives an email summarizing the request fulfillment: it includes the server’s name, IP address, credentials or access info (e.g., “this Linux server has been added to domain X, you can SSH with your AD account” or “the default local admin password is… if applicable”), and links to documentation or next steps. If there are any **post-provision manual steps** (for example, a requirement to register the server in a monitoring tool or load balancer), the workflow can generate follow-up tasks to the responsible teams as part of closure. At this point, from the end user’s perspective, their request has been fulfilled – they have the new server they asked for, and all details are recorded.

The ServiceNow request item (RITM) is then marked **Completed**. Every activity that occurred (approvals, IP addresses assigned, VM details, etc.) is captured in the work notes for traceability. The user can always go back to this record to see the full history of how their request went from submission to completion.

### Exception Handling and Logging

**Step 10: Exception Handling:** The workflow is built to handle exceptions gracefully at each critical step[[27]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=6,It%20will%20then)[[36]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=nm%20,keep%20the%20environment%20clean). If any integration fails or a condition is not met, the system will not simply crash silently. For example: - If the IP management system IP allocation step fails (no IP available or API timeout), the workflow will **stop before provisioning** and mark the request as needing admin attention. It might generate a ServiceNow incident or task to the network team to resolve the IP issue, and notify the requester of a delay.  
- If the automation platform provisioning returns an error, the workflow immediately flags the request as **Failed**. A notification is sent to the requester stating that the build encountered an error and the IT team is addressing it. Simultaneously, an incident can be created or the failure is assigned to the appropriate support queue with the error details (e.g., “vCenter cluster out of resources – VM not created”). In such a case, the workflow will also perform **rollback** actions: for instance, if an IP was reserved but the VM wasn’t created, it will call IP management system to release that IP and remove the DNS record[[36]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=nm%20,keep%20the%20environment%20clean), so there are no orphaned allocations.  
- If an approval is not responded to in time (SLA breached) or an approver rejects with an edit needed, the workflow can cancel or pause accordingly, rather than proceeding incompletely.

All these branches ensure that when something goes wrong, it’s managed transparently and doesn’t leave the system in an inconsistent state. The users are kept informed, and IT teams have clear triggers to jump in. This is a big improvement over the manual process, where failures could go unnoticed until someone manually checked status.

**Step 11: Logging and Auditing:** Throughout the process, **detailed logging** is implemented for audit and debug purposes[[37]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=7,is%20useful%20for%20compliance%20reviews). Every integration call (to IP management system, automation platform, etc.) and every major decision (skipping an approval, etc.) writes a log entry either in ServiceNow’s workflow context or a custom log table. For example, when an IP is reserved, the log notes “Reserved IP 10.14.20.57 from IP management system for request RITM0012345”. When automation platform deployment starts, it logs the request ID and when it finishes, it logs success/failure and timing. These logs create an **audit trail** proving that governance was followed (e.g., one can see all approval timestamps and who approved, which is useful for SOX compliance or internal audits). They also greatly aid in troubleshooting – if a request didn’t complete as expected, we can see exactly which step failed and why. All approvals and task outcomes are inherently logged by ServiceNow as well. We will also leverage ServiceNow’s reporting to generate metrics from these logs (like average provisioning time, any repeated failure points, etc.) to continuously improve the process.

In essence, the technical workflow is designed to be **robust and transparent** – coordinating human and machine steps seamlessly. By the time a single request flows through this pipeline, it will have either delivered a fully configured server or clearly reported why it could not, with all intermediate steps documented. This is a stark contrast to the current state where one has to chase different teams for updates. The automation not only speeds things up, but also provides clarity and accountability at each stage.

## Integration Architecture Diagrams and Flowcharts

To visualize the solution, this section includes key architecture diagrams and flowcharts that illustrate how components interact and how the process flows:

**High-Level Integration Architecture:** The following diagram shows how ServiceNow, acting as the central orchestrator, connects with external systems to fulfill a server request. ServiceNow communicates outbound through a to both IP management system (for IP/DNS) and to automation platform (for VM provisioning). automation platform in turn interfaces with vCenter to create the VM, and updates are fed back to ServiceNow for CMDB and notification purposes. All user interaction is through ServiceNow’s portal.

*(See embedded Figure in Section 4 above for high-level architecture.)*

**Workflow Swimlane Diagram:** *(Placeholder – to be included)* A swimlane diagram will be used to depict the end-to-end process across different roles/systems. For example, lanes for Requester, ServiceNow, Approval Teams, Network (IP management system), automation platform/vCenter, and Ops, showing how the request transitions from one to the other. This complements the flowchart by emphasizing responsibility hand-offs – e.g., ServiceNow handles steps X through Y, automation platform handles the VM creation sub-process, etc.

**Provisioning Flowchart:** The flowchart embedded below (and also in Section 6) details the automated workflow logic from request submission to completion. It highlights decision points (diamonds) such as whether certain approvals are needed and actions (rectangles) like “Reserve IP” or “Deploy VM”. This diagram is useful for both developers (to implement the flow) and reviewers (to understand the sequence at a glance).

*(Flowchart image is embedded in Section 6 above as Figure: Automated provisioning workflow.)*

In addition to these, separate **sequence diagrams** may be prepared for specific integrations – for instance, a sequence diagram showing the step-by-step API calls when provisioning a VM: ServiceNow sends API call to automation platform, automation platform provisions VM, automation platform calls back or ServiceNow polls, etc., including IP management system interactions. These diagrams provide a low-level view of the timing and invocation of each API across systems and will be included in the appendix for technical reference.

By using these visual artifacts, stakeholders can clearly see how data and control flow through the new system, and engineers have a blueprint for where and how to build integration points.

## User Stories and Agile Implementation Plan

To implement this solution, we will follow an **Agile approach**, breaking the work into epics and user stories delivered over multiple sprints. This ensures incremental progress, early feedback, and the ability to adapt as needed. Below is an overview of the planned **Epics** (major functional components) and key User Stories within each:

* **Epic 1: ServiceNow Catalog & Front-End** – *Goal:* Create the user interface and basic workflow in ServiceNow for server requests.
* *Story 1.1:* **Catalog Form Design** – Develop the “New Server Request” catalog item with all required fields (project info, server specs, etc.) and validate form behavior. *Acceptance:* Users can submit a request with all necessary data captured.
* *Story 1.2:* **Dynamic Fields & Naming** – Implement dynamic field logic (e.g., conditional visibility for AIX-specific inputs) and auto-generation of hostnames based on naming conventions. *Acceptance:* Hostnames auto-populate correctly; irrelevant fields are hidden for certain choices.
* *Story 1.3:* **Initial Workflow & Record Generation** – Configure a basic flow that creates a request record (RITM) upon submission and maybe sends a confirmation. This is the scaffold for adding approvals and integration later.
* **Epic 2: VMware vRealize Automation (automation platform) Integration & Cloud Template** – *Goal:* Enable automated VM provisioning through automation platform.
* *Story 2.1:* **Cloud Template (Blueprint) Development** – In automation platform, create a reusable blueprint for a standard VM build (parameterized by CPU, RAM, OS image, etc.). *Acceptance:* The blueprint can deploy a VM in automation platform (tested within automation platform using sample inputs).
* *Story 2.2:* **automation platform Configuration for Environments** – Set up automation platform Project, cloud zones, image mappings, and network profiles corresponding to Dev, QA, Prod environments so the blueprint deploys to correct resources.
* Story 2.3: ServiceNow ↔ provisioning system connection – Enable the existing integration pathway as agreed with the automation team in ServiceNow or develop REST integration. Securely store automation platform credentials/tokens in SN. Acceptance: ServiceNow can trigger a deployment in automation platform (e.g., a test API call results in a VM being created).
* *Story 2.4:* **End-to-End automation platform Provisioning Test** – Tie it together: when a SN request is approved and reaches the provisioning step, ensure a VM actually gets created in vCenter via automation platform and the status is reported back.
* **Epic 3: IP management system IPAM & DNS Automation** – *Goal:* Automate IP address assignment and DNS registration within the workflow.
* *Story 3.1:*  **Prep** – Ensure a ServiceNow with network access to IP management system is up and running. *Acceptance:* can reach IP management system API (validated by a test ping or API call).
* *Story 3.2:* **Credential Management** – Store IP management system API credentials in SN’s credential vault or IntegrationHub connection. No hard-coded secrets.
* Story 3.3: IP Reservation Script/Action – ServiceNow uses the existing process to retrieve the next available IP from the network automation system. Acceptance: Given a subnet ID, the workflow retrieves an IP (tested in a lower env or dummy network).
* *Story 3.4:* **DNS Registration** – Develop action to create a DNS host record in IP management system with hostname and IP[[2]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=Story%203,name). *Acceptance:* After running, the hostname resolves to the IP in DNS.
* *Story 3.5:* **Workflow Integration** – Insert IPAM steps into the SN flow at the right point (after approvals, before automation platform). Handle responses and pass the acquired IP to the automation platform provisioning step.
* *Story 3.6:* **Error/Rollback Logic** – Implement logic to handle IP management system failures (e.g., no IP available -> notify network team, pause workflow) and to release IP if provisioning fails later[[38]](file://file-J2VboKwe925g7NWpEaNUeD#:~:text=around%20the%20IPAM%20steps%20,These%20rollback%20steps%20can%20be).
* **Epic 4: Orchestration Workflow & Approvals** – *Goal:* Build the complete ServiceNow flow that ties all steps (approvals, integrations, notifications) together.
* *Story 4.1:* **Approval Workflow Configuration** – Define the approval rules in the flow (Architecture, then conditional others). Use Flow Designer or Workflow with decision branches for each condition. *Acceptance:* A submitted request routes to the correct approvers and waits for their response in order.
* *Story 4.2:* **Parallel Approval Optimization** – If using Flow Designer, configure it to request multiple approvals in parallel when applicable (e.g., Network and Security at the same time after Arch approval). Ensure it still waits for all to finish.
* *Story 4.3:* **Integration Steps in Flow** – Insert the IP management system integration step and automation platform trigger step into the workflow in sequence, with proper data flow between them (e.g., IP from IP management system feeds into automation platform call).
* *Story 4.4:* **Notification & Closure** – Add steps at the end of the flow to send completion emails and update the RITM/Change request state to “Completed”.
* *Story 4.5:* **Failure Paths** – Implement the branching for any failure: if IP management system or automation platform integration returns an error flag, update the request state to “Failed” or “On Hold”, notify support teams (perhaps auto-create an incident), and inform the requester of next steps.
* **Epic 5: CMDB and Asset Management Integration** – *Goal:* Ensure assets are tracked in CMDB and existing processes are updated.
* *Story 5.1:* **CI Creation** – Determine the method for CMDB update (via automation platform plugin sync vs. direct SN update). Implement creating a new CI in SN CMDB upon successful VM build with the data we have. *Acceptance:* After a test deployment, a corresponding CI is visible in CMDB with correct attributes.
* *Story 5.2:* **Post-Provision Discovery** – Configure a targeted SN Discovery or orchestration to run on the new server to gather detailed info (e.g., schedule a quick discovery or use an agent if available).
* *Story 5.3:* **Audit and Reconcile** – Ensure that if automation platform auto-inserts a CI as well, we reconcile duplicates or use one approach consistently to avoid double entries.
* *Story 5.4:* **Integration with Change/Incident** – (Optional) Make sure the provisioning workflow ties into our Change Management process if required (we might have a parent change record for each build for audit). Also ensure any failures create incidents as needed.
* **Epic 6: Security, Testing, and Deployment** – *Goal:* Validate the solution end-to-end, address security requirements (especially for DMZ), and roll out to production.
* *Story 6.1:* **End-to-End Test Cases** – Develop test scenarios covering common requests (Win/Lin dev server, Prod high-classification server, AIX server, etc.) and run through the workflow in a non-prod environment. Verify all steps execute correctly and the server comes out as expected.
* *Story 6.2:* **Performance Testing for Batch Requests** – If the form allows requesting multiple servers in one submission, test with, say, 5-10 server requests at once to ensure the workflow and automation platform can handle provisioning concurrently and within SLA.
* *Story 6.3:* **Security Review** – Conduct a review with our Security Architecture team: confirm that the and integration accounts follow all policies, the DMZ access contract is in place and tested (attempt a sample DMZ VM deployment to verify network connectivity on required ports only), and that logging/auditing meets requirements.
* *Story 6.4:* **User Training & UAT** – Provide training or demos to key user groups (e.g., the operations team, application teams that frequently request servers) and run a User Acceptance Testing phase where a few pilot requests are done by end-users with oversight. Incorporate feedback (e.g., if form questions are unclear or additional fields are needed).
* *Story 6.5:* **Production Deployment** – Plan the go-live of the new workflow. This involves scheduling a ServiceNow update set deployment, any automation platform blueprint promotions, and ensuring IP management system integration is pointing to production IP management system. Also communicate the change to the IT teams (so they know to expect automated requests instead of emails). Monitor the first few production requests closely and be ready to fall back to manual process if any critical issues arise.

Each epic above corresponds to a logical component of the solution. We anticipate delivering this over roughly **3–4 sprints (approx. 8–12 weeks)**. For example, Sprint 1 might tackle Epic 1 and parts of Epic 4 (basic form and single approval workflow), Sprint 2 focuses on Epic 2 (automation platform integration) and Epic 3 (IP management system integration), Sprint 3 finalizes Epic 4 (full workflow with all branches) and Epic 5 (CMDB), and Sprint 4 concentrates on Epic 6 (testing, security, rollout). Throughout, we will engage stakeholders for feedback – e.g., demo the form and approvals after Sprint 1, do an early provisioning test in Sprint 2 to show a VM getting created automatically, etc. Using this agile and iterative approach will help ensure the final product meets expectations and any issues are ironed out before full deployment.

## System Security and Access Controls

Security and access control are paramount in this solution, as it involves automated actions across various systems and network zones. The design incorporates multiple layers of security controls:

* **Role-Based Access in ServiceNow:** The ServiceNow catalog item and workflow will enforce proper **access controls**. Only authorized users (e.g., certain user groups or roles) can submit server requests for certain environments (for example, only IT staff can request Prod servers). Approvals are routed to specific groups, and only designated approvers can approve their respective tasks. Within ServiceNow, sensitive fields (like data classification) can be made read-only or hidden from unauthorized viewers. The workflow and scripts run under a service account with elevated rights, but normal users cannot bypass the process or trigger back-end actions without following the approval chain.
* **Integration Credentials and Permissions:** Integration accounts for automation platform and IP management system are **limited-privilege accounts**. For automation platform, we will use an API service user that can deploy VMs and read templates, but cannot, say, delete resources arbitrarily or modify automation platform configuration. For IP management system, the API user will have permission just to reserve IPs and create DNS records in specific DNS zones. All credentials for these accounts are stored securely in ServiceNow (encrypted) and/or the host’s credential store – never in plain text in scripts. The itself runs under a service account with only the necessary network access. API communications use HTTPS and will leverage certificate-based authentication or token-based auth where possible to avoid password usage.
* **Network Security and DMZ Controls:** As described in Section 5 (Secure DMZ Access), strict network controls govern any automation traffic. The ServiceNow and automation platform appliance will communicate with *internal* vCenter and other internal systems freely on necessary ports, but for any **DMZ-located server provisioning**, only the specific allowed ports (22, 5986, 443, etc.) are open from the automation sources to the DMZ destination. This principle of **least network access** ensures that even if the automation systems were compromised, they cannot open arbitrary connections or transfer data from the DMZ back into internal networks (no inbound). We have aligned these rules with FRB’s zero-trust stance for the DMZ[[39]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=No%20automation%20or%20orchestration%20tools,from%20the%20secure%20internal%20zone)[[7]](file://file-2DPqsZDjGdW5Sk9pguTrHb#:~:text=Create%20explicit%20ACI%20contract%3A%20Allow,Automation%20EPG%20to%20DMZ%20EPG). In addition, any configuration management (like configuration management tool playbooks) that run on target servers are executed over these approved channels and use ephemeral session credentials – there are no permanent agents on DMZ servers listening for instructions, further reducing attack surface.
* **Audit Trail and Logging:** All provisioning actions and approvals are **logged in detail**. ServiceNow provides a complete audit trail of who approved what and when, and we are augmenting this with integration logs (each API call outcome is recorded). Changes made in external systems (e.g., IP management system entries created or vCenter VMs created) can be cross-audited with these logs. The also keeps a log of all commands it executes. For critical changes in sensitive zones (like any contract changes for DMZ access), those go through the formal change management process with Security oversight. We will retain logs according to policy (e.g., keep provisioning logs for 1 year or as required by audit). These logs facilitate periodic compliance reviews to verify that, for instance, every production server had a corresponding approved request and security review.
* **Separation of Duties:** The solution is implemented in a way that respects separation of duties. No single person can request and unilaterally approve a server build in production – different teams must concur (enforced by the workflow). The administration of the ServiceNow workflow and the operation of automation platform/IP management system are done by different teams, adding checks and balances. For example, even though the process is automated, the Security team’s approval step ensures they retain oversight on high-risk builds. Additionally, the credentials used by automation are separate from those used by individuals, and their use is restricted to the automated context (and monitored).
* **Secure Development & Testing:** The implementation itself will be done following secure SDLC practices. That includes code review of any scripts (to avoid vulnerabilities like hard-coded creds or injection flaws), testing in non-production environments, and ensuring the ServiceNow instance and s are up-to-date on patches. ServiceNow and automation platform will be configured with proper TLS certificates. The IP management system and automation platform endpoints are internal and secured by firewall; the will initiate connections such that no external exposure is needed.
* **Data Handling and Privacy:** The information captured in the request (like server purpose, possibly some application data) will be treated as internal confidential. We will ensure the ServiceNow form and underlying tables have appropriate access control so that only relevant IT staff can see potentially sensitive details (for example, if someone puts a regulatory project name or high data classification, not everyone on the platform should see it except those involved).
* **Exception and Alerting:** If there are multiple failed provisioning attempts or any security-related exceptions (like an unexpected denial from IP management system or automation platform), the system can alert administrators. We can integrate with a SIEM or monitoring tool to flag unusual events (for instance, if someone somehow triggers an out-of-band change in the process, or if the tries to execute an unknown command).

In conclusion, security has been baked into the architecture from the ground up – from network restrictions and credential management to approval gates and audit trails. This ensures the automation delivers speed and efficiency **without** compromising on the control and security requirements our organization, and regulators, expect.

## Collaboration and Governance Framework

Successful implementation and ongoing operation of this solution require close collaboration among multiple teams and a clear governance structure. Below are the key elements of the collaboration and governance framework:

* **Cross-Functional Project Team:** During the build-out, we will maintain a project team comprising members of the **ServiceNow team**, the **Automation/DevOps team** (for automation platform and configuration management tool), the **Network team** (for IP management system and network policy input), **Security** (to advise on controls, especially DMZ-related), **Platform/Server Ops** (for VMware and AIX expertise), and **Application reps** (for user perspective). This ensures all perspectives are considered. Regular scrum meetings and bi-weekly demos to stakeholders will keep everyone aligned and able to provide input continuously.
* **Stakeholder Steering Committee:** For governance at a management level, an **Architecture & Automation Steering Committee** will oversee the project. This might include the infrastructure architecture lead, operations managers, and IT risk/compliance representatives. They will meet (perhaps monthly) to review progress, approve major design decisions (e.g., any scope changes or policy exceptions), and clear roadblocks. Having this body helps maintain strategic alignment and executive support.
* **Communication and Training:** We will engage end-user representatives (like a few power users who frequently request servers) early on to get feedback on the catalog design. Prior to go-live, the team will prepare **user guides** and offer training sessions or “lunch and learns” for IT staff and requester groups. Clear communication will be sent out about the new process, how to use it, and whom to contact for support. This avoids confusion and encourages adoption.
* **Governance Policies:** Post-implementation, the solution will be governed by existing ITSM and DevOps processes. For instance:
* **Change Management:** Changes to the workflow (after go-live) will go through CAB review as they potentially impact multiple teams. Similarly, enabling new features (like adding a new site/subnet mapping) will involve notification to stakeholders.
* **Approval Matrix Maintenance:** A designated owner (e.g., the ServiceNow platform team in coordination with the Architecture team) will periodically review the approval rules and groups. If organizational changes occur (team names, responsibilities), the workflow’s approval routing will be updated accordingly to ensure it stays current.
* **Continuous Improvement:** After rollout, we will establish a feedback loop. The automation team will gather metrics (e.g., average provisioning time, any recurring errors, or bottlenecks) and also solicit feedback from users and approvers. This could be via a quarterly review meeting or a feedback form on the ServiceNow item. The team will prioritize enhancements in a backlog for future sprints. This might include adding new catalog options, further reducing manual steps, or optimizing performance.
* **Documentation and Knowledge Sharing:** All aspects of the solution will be documented – from architecture diagrams to user story definitions to runbooks for operations. An internal knowledge base will be updated so that, for example, support teams know how to troubleshoot a failed request (with steps to manually fulfill if needed). We will also document roles and responsibilities: e.g., the ServiceNow team owns the workflow logic, the automation platform team owns the cloud template and underlying virtualization environment, the network team owns IP management system and IP space management, etc. This clarity ensures everyone knows their part in supporting the system.
* **Operational Handoff and Support:** Before going live, an operational readiness review will be done. The support model likely involves the ServiceNow/platform team as L2 support for any workflow issues, and respective domain teams (VMware, Network) as L3 for issues in their domains (like IP management system errors or vCenter capacity issues). We will set up an on-call rotation if needed for the initial period after launch to quickly address any incidents. Governance here means no single team is overburdened; responsibilities are shared and clear.
* **Governance of Future Enhancements:** A governance board (could be the same steering committee or a subset) will evaluate major future changes or extensions to the system. For example, if in six months we propose to integrate AWS or Azure VMs into this ServiceNow workflow, that would be reviewed and approved through governance to ensure it fits the enterprise architecture.

By fostering collaboration during the project and establishing governance for ongoing changes, we ensure the solution not only gets delivered successfully but is sustainable and continuously improved upon. This partnership between Architecture, Operations, Security, and other stakeholders is a core strength of the initiative – it turns what was previously a fragmented process into a shared, well-managed service.

## Future Enhancements Roadmap

While this document focuses on the current scope (ServiceNow + automation platform + IP management system for VMware and AIX provisioning), there are several opportunities to extend and enhance the solution after the initial implementation. Our roadmap for future enhancements includes:

* **Automated AIX LPAR Provisioning:** In the initial phase, AIX provisioning may still involve manual steps or basic automation. A key next step is to implement full **API-driven AIX LPAR provisioning**. This could involve integrating IBM PowerVC (if adopted) with ServiceNow so that AIX LPARs are created just as automatically as VMs, or expanding configuration management tool playbooks to handle end-to-end LPAR creation through the HMC. Achieving parity for AIX with the VMware automation will further reduce manual work and allow AIX teams to focus on higher-level tasks.
* **Physical Server Automation:** Extend the framework to cover physical server requests (if they remain part of our infrastructure strategy). For example, integrate with out-of-band management tools (like HPE iLO or Dell iDRAC APIs) or with Data Center Infrastructure Management (DCIM) software to automate steps in physical provisioning (like booting from network, etc.). At minimum, use the workflow to track and manage physical builds with tasks, and explore partial automation such as using configuration management tool for OS deployment on bare metal.
* **Multi-Cloud Provisioning:** Adapt the ServiceNow catalog and workflow to provision resources in public clouds (AWS, Azure, etc.) using a similar model. This might be via automation platform’s multi-cloud capabilities or direct IntegrationHub spokes for cloud services. For example, a future story could allow a user to request “AWS EC2 instances” through the same interface, with ServiceNow calling AWS CloudFormation or Terraform in the backend. The architecture is positioned as a **single pane of glass** for hybrid cloud provisioning, so incorporating cloud endpoints would be a natural evolution.
* **Infrastructure as Code Integration:** Leverage Infrastructure-as-Code (IaC) pipelines such that the ServiceNow requests can generate IaC artifacts or tie into version-controlled templates. For instance, the VM blueprint definitions in automation platform could be managed as code (YAML in Git) and the ServiceNow workflow triggered via a pipeline. Similarly, explore using Terraform Enterprise integration with ServiceNow for provisioning as code. This would improve maintainability and allow infrastructure definitions to be treated like software code with reviews and versioning.
* **Enhanced Configuration Management:** Introduce a **Configuration Management** phase post-provisioning. For example, integrate with configuration management tool Tower (AWX/AAP) or Puppet to automatically apply standard configurations to new servers (beyond what automation platform does). This could include installing a standard software agent suite, performing security baseline configurations, or registering the server with monitoring and backup systems. Currently, we send notifications or tasks for some of these – in the future, they could be automated steps for even more touchless provisioning.
* **Self-Service Improvements:** Over time, refine the ServiceNow portal experience. This could include **Blueprint Selection** (allow users to choose from predefined server types or stacks), cost estimation (show an estimated cost for the requested resources before submission), and a **knowledge base integration** (display relevant architecture standards or tips inline on the form). We might also allow cloning of previous requests for convenience if a user needs to request similar servers repeatedly.
* **Analytics and Optimization:** Implement dashboards and reports in ServiceNow to track metrics like provisioning time, approval durations, failure rates, etc. Use this data to identify where further improvements can be made (e.g., a consistently slow approval could indicate a need for policy change or more approvers). We can also integrate with an AI Ops tool in the future to predict capacity needs – for instance, if many servers of a type are being requested, proactively alert capacity planners.
* **Chatbot and API interfaces:** Enable a chatbot interface or API for requests – for example, allow users to request a server via Slack/MS Teams using a chatbot that interacts with ServiceNow, or through a DevOps pipeline where a pipeline script calls the ServiceNow API to create a request programmatically as part of application deployment. This would further streamline the developer experience.
* **Continuous Compliance and Audit Integration:** As we move more into automation, we can enhance compliance checks – e.g., automatically verify after provisioning that server configurations meet policies (perhaps by running a compliance scan or checking CIS benchmarks via tools integrated with configuration management tool). We could also integrate with GRC (Governance, Risk, Compliance) modules in ServiceNow to log each new server for any required compliance attestations.

Each of these enhancements will be evaluated and prioritized based on business needs and resource availability. The architecture put in place now is intentionally scalable to accommodate these changes. By following an agile, iterative improvement approach, we will continue to modernize and expand the automation capabilities, maintaining our momentum towards a fully autonomous infrastructure provisioning service.

## Appendices

### A. JSON Schema – ServiceNow Catalog Item Example

For reference, the snippet below illustrates a portion of the JSON definition for the ServiceNow catalog item (as exported via ServiceNow’s export functionality). It defines the variables (fields) used in the form, their types, and some of the choices. This schema ensures consistency in data capture and can be used for auditing or migrating the catalog item between instances.

{  
 "name": "Virtual Server Request",  
 "description": "Submit a virtual server build request with automated approvals and provisioning integration.",  
 "variables": [  
 { "name": "project\_number", "type": "string", "mandatory": true },  
 { "name": "project\_name", "type": "string", "mandatory": true },  
 { "name": "project\_manager","type": "reference", "mandatory": true, "reference\_table": "sys\_user" },  
 { "name": "os\_type", "type": "choice", "mandatory": true, "choices": ["Windows", "Linux", "AIX"] },  
 { "name": "build\_type", "type": "choice", "mandatory": true, "choices": ["New", "Rebuild", "Replatform"] },  
 { "name": "virtual\_or\_physical", "type": "choice", "mandatory": true, "choices": ["Virtual", "Physical"] },  
 { "name": "cpu\_cores", "type": "integer", "mandatory": true },  
 { "name": "ram\_gb", "type": "integer", "mandatory": true },  
 { "name": "site\_location", "type": "choice", "mandatory": true, "choices": ["NJDC1", "AZDC2", "AWS-us-east-1"] },  
 { "name": "network\_tier", "type": "choice", "mandatory": true, "choices": ["DMZ", "App", "DB"] },  
 { "name": "ip\_address", "type": "string", "mandatory": false },  
 { "name": "hostname", "type": "string", "mandatory": false, "readonly": true },  
 ...  
 ],  
 "workflow": "u\_virtual\_server\_fulfillment\_workflow"  
}

*Example: JSON excerpt defining catalog item variables in ServiceNow. Many fields are choice lists to ensure data standardization. The workflow property points to the associated fulfillment workflow.*

### B. Hostname Generator Script (Excerpt)

The automated hostname generation follows a strict 16-character format composed of multiple segments (site, region, app acronym, environment, function, sequence, etc.). Below is an excerpt from the Python script used to generate hostnames, which could be utilized in a ServiceNow server-side script or run offline for bulk name generation:

def generate\_hostname(site\_type, region\_id, ciz\_id, tenant, app\_acronym, environment, function, sequence):  
 site\_type = site\_type.upper() # e.g., "Z" for primary DC  
 ciz\_id = ciz\_id.upper() # e.g. "A" for zone A  
 tenant = tenant.upper()[:2] # up to 2 chars  
 app\_acronym= app\_acronym.upper()[:4] # up to 4 chars  
 environment= environment.upper()[:3] # up to 3 chars (DEV, QA, PRD, etc.)  
 function = function.upper()[:2] # up to 2 chars (AP for app, DB for db, etc.)  
 seq = str(sequence).zfill(2) # zero-padded 2-digit sequence  
 hostname = f"{site\_type}{region\_id}{ciz\_id}{tenant}{app\_acronym}{environment}{function}{seq}"  
 if len(hostname) > 16:  
 hostname = hostname[:16] # ensure 16 char limit  
 return hostname  
  
# Example usage:  
print(generate\_hostname("Z", "2", "A", "SH", "IBLX", "PRD", "AP", 1))  
# Output: "Z2ASHIBLXPRDAP01"

This logic is incorporated into the ServiceNow workflow to generate hostnames automatically. Hostnames are constructed to be unique and encode key information (in this example, “Z2” might indicate Data Center 2, “SH” a business tenant code, “IBLX” an application code, “PRD” environment, “AP” function, and “01” sequence). The script ensures names do not exceed the 16-character limit and truncates with a warning if so.

### C. Sequence Diagram – API Interactions (ServiceNow, , automation platform, IP management system)

*(This section will include a diagram or description of the sequence of API calls and responses between systems during the provisioning process.)* For example, the sequence for a successful VM provision might be:

1. **ServiceNow → (IP management system):** *GET next\_available\_ip* – ServiceNow requests an IP via the , IP management system responds with IP X.
2. **ServiceNow → (IP management system):** *POST create host record* – ServiceNow sends hostname Y and IP X to IP management system to create DNS entry. Response indicates success.
3. **ServiceNow → (automation platform):** *POST /deployments* – ServiceNow calls automation platform API (or plugin triggers it) to deploy blueprint with params (including IP X, hostname Y, etc.). automation platform returns a Request ID.
4. **ServiceNow (flow)** – waits/polls.
5. **automation platform → vCenter:** (internal) automation platform orchestrates VM creation on vCenter.
6. **automation platform → ServiceNow:** *callback or status poll response* – indicates VM is deployed (or plugin updates the record).
7. **ServiceNow → (CMDB):** *PUT CI record* – SN creates/updates the CMDB CI for the new VM.
8. **ServiceNow:** *Send notifications* – SN notifies requester and others of completion.

In case of errors, alternate paths include IP management system returning “no available IP” or automation platform returning a failure, upon which ServiceNow logs the error and notifies IT support. The exact sequence diagram will be provided to visually map these interactions, showing the order and dependency of each integration call.

### D. API Field Definitions and Integration Payloads

This section provides examples of key API payloads and field mappings used in integration:

* **IP management system API (WAPI) Payload:** For reserving an IP and creating DNS: a typical JSON payload to create a host record might look like:
* {  
   "name": "servername01.corp.example.com",  
   "ipv4addrs": [  
   {  
   "ipv4addr": "10.14.20.57"  
   }  
   ]  
  }
* When POSTed to the IP management system /record:host endpoint, this simultaneously reserves the IP 10.14.20.57 and creates a DNS A record mapping the hostname. The ServiceNow workflow constructs this JSON using the hostname it generated and the IP from the earlier call.
* **automation platform API Payload:** If using direct REST integration with automation platform 8, the ServiceNow script might POST a JSON to https://vra.company.com/codestream/api/blueprints/<blueprintID>/deployments (example endpoint). The payload would include inputs such as:
* {  
   "deploymentName": "servername01",  
   "inputs": {  
   "cpu": 4,  
   "memory": 16,  
   "ipAddress": "10.14.20.57",  
   "osType": "RHEL8",  
   "environment": "Prod"  
   }  
  }
* These inputs correspond to the blueprint’s parameters (defined in automation platform’s Cloud Template). The blueprint ID itself is referenced (or if using the plugin, ServiceNow just calls the catalog item with these as variables).
* **CMDB Fields Mapping:** The new server CI in ServiceNow CMDB will have fields populated from the request:
* *Name* = Hostname (e.g., servername01)
* *IP Address* = 10.14.20.57 (primary IP)
* *Environment* = Production (from the request field)
* *OS* = Red Hat Enterprise Linux 8 (from OS selection)
* *CPU, RAM* = as requested (e.g., 4 vCPU, 16 GB)
* *Application* (if provided) = linked to corresponding Business Application CI
* *Owner* = Requester or Project Manager (depending on how we map it)
* *Virtual Cluster* = references the vCenter cluster (if we integrate with vCenter data)
* *Serial Number/VM ID* = updated later via discovery or automation platform sync

An example of using ServiceNow’s Table API to create the CI record might involve a JSON like:

{  
 "name": "servername01",  
 "ip\_address": "10.14.20.57",  
 "os\_version": "Red Hat Enterprise Linux 8",  
 "cpu\_count": 4,  
 "ram": 16,  
 "u\_environment": "Production",  
 "u\_application": "Finance Trading App",  
 "owned\_by": "david.smith"  
}

posted to the cmdb\_ci\_server table API. (The exact field names differ in our instance and some values might be reference links.)

Providing these examples in the appendix helps future developers or integrators understand how the pieces communicate and can serve as a reference when troubleshooting or extending integrations.

### E. Visual Workflow & Swimlane Diagrams

*(Placeholder for any additional diagrams.)* In addition to the flowchart provided, any complex logic can be further broken down. For instance, a **swimlane diagram** as mentioned can clarify which team or system is responsible for which part of the process. We will include such diagrams to aid clarity: e.g., showing ServiceNow actions vs. automation platform actions vs. user actions on a timeline.

### F. Reference Links and Documentation

* **VMware vRealize Automation 8.x Documentation:** Official docs on blueprint creation, API usage, and the ITSM integration plugin. (e.g., VMware Docs link for automation platform 8 API and plugin setup)
* **IP management system NIOS API (WAPI) Documentation:** Reference for IP management system REST API calls used for IP address and DNS management.
* **ServiceNow IntegrationHub Spokes:** Documentation for any pre-built spokes used (IP management system IPAM spoke, automation platform plugin documentation from ServiceNow store).
* **FRB DMZ Security Policy:** Internal security policy document outlining requirements for segmentation and access (used to design the DMZ access solution).
* **Naming Convention Standard:** Internal document defining the server naming scheme (to which our hostname script adheres).
* **ServiceNow CMDB Data Model:** Reference to which CI classes/fields we use for servers, to ensure compliance with CMDB standards.  
  *(These references should be consulted for detailed parameters and are useful for auditors or new team members to understand external constraints and configurations.)*

### G. Glossary of Terms

* **AIX:** IBM’s Unix operating system (Advanced Interactive eXecutive) typically run on IBM Power systems.
* **configuration management tool:** An automation tool used for configuration management and application deployment, known for its agentless architecture (uses SSH/WinRM). In our context, used to automate AIX tasks and post-provision configurations.
* **CMDB:** Configuration Management Database, a repository in ServiceNow that stores information about hardware and software assets (Configuration Items) and their relationships.
* **DDI / IPAM:** Stands for DNS, DHCP, and IP Address Management – IP management system DDI is the system managing these network services, often simply referred to as IPAM (IP Address Management).
* **DMZ:** Demilitarized Zone, a network segment isolated from internal networks, usually hosting externally-facing systems. Subject to stricter access controls.
* **:** Management, Instrumentation, and Discovery Server – a lightweight ServiceNow agent installed on-premises that facilitates communication between ServiceNow (cloud) and on-prem systems (like automation platform, IP management system), bypassing firewall restrictions.
* **automation platform (vRealize Automation):** VMware’s automation platform for provisioning and managing infrastructure (VMs, networks, etc.) across hybrid cloud environments. We use it to automate VMware vSphere VM deployments.
* **vCenter:** VMware vCenter Server, the management hub for vSphere environments. automation platform interfaces with vCenter to perform the actual VM cloning and creation operations.
* **WAPI:** Web API – IP management system’s REST API interface for automation tasks.
* **RITM:** Request Item – in ServiceNow ITSM, each catalog request generates a RITM record that tracks the fulfillment of that specific item. In our case, the “New Server” request corresponds to a RITM which the workflow manages.
* **LPAR:** Logical Partition – a virtualized instance of AIX running on an IBM Power physical server. Similar to a VM concept but in the IBM Power environment.
* **PowerVC:** IBM Power Virtualization Center – a tool for cloud-like management of IBM Power systems (including AIX VMs), providing APIs for provisioning which could be integrated in the future.

*End of Document – The above architecture design provides a comprehensive view of the proposed automated server provisioning solution, including the current challenges it addresses, the target state design and components, detailed workflow, and considerations for implementation and beyond. It is intended to guide the implementation team and inform stakeholders of how the solution will function and be governed.*