

# Review on Audit-Ready System Builds Using SMF and Puppet

Kateryna Holub, Oleksandr Kravchuk, Natalia Koval, Yuriy Sydorenko  
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine

**Abstract-** In regulated IT environments, achieving audit-ready system builds is crucial for maintaining compliance, operational integrity, and trust. This review explores how the integration of Solaris Service Management Facility (SMF) and Puppet configuration management enables the creation of infrastructure that is both resilient and verifiable. SMF offers deterministic service lifecycle control, dependency resolution, and fault recovery, while Puppet ensures declarative system provisioning, configuration drift correction, and policy enforcement. Together, they form a robust framework for building and maintaining UNIX systems that meet stringent compliance standards such as HIPAA, SOX, PCI-DSS, and ISO 27001. This article details integration patterns between Puppet and SMF, including automated service registration, state enforcement, and logging strategies that support continuous compliance verification. Real-world use cases from healthcare, finance, and scientific research sectors highlight the scalability and traceability benefits of this approach. Further, the paper addresses challenges in manifest maintenance, performance bottlenecks, and error debugging, offering practical mitigation strategies. Emerging trends such as Policy-as-Code, AIOps integration, and immutable infrastructure are also discussed, illustrating the direction of future-ready, compliance-driven automation. By aligning infrastructure-as-code principles with service-level orchestration, this framework transforms audit-readiness from a reactive task into a continuous, automated operational model.

**Index Terms-** Audit Readiness, Solaris SMF, Puppet, Infrastructure as Code, Compliance Automation, Configuration Management, Declarative Systems, System Integrity, Policy-as-Code, Zero Trust, Immutable Infrastructure, Regulated Environments, UNIX Automation, Service Lifecycle Management, Fault Recovery, Compliance-as-Code

## I. INTRODUCTION

### 1. Audit-Ready System Philosophy

An audit-ready system is architected with the principle of visibility, reproducibility, and traceability at its core. In highly regulated environments such as healthcare, finance, and defense systems must demonstrate that every aspect of their configuration and operation adheres to internal policy and external compliance standards like HIPAA, SOX, or ISO 27001. This means systems must be consistently configured, changes must be logged and justifiable, and the system must be restorable to a known compliant state at any time. Audit-ready builds emphasize immutability, policy enforcement, and continuous monitoring, ensuring that operational behavior aligns with governance requirements.

### 2. Relevance of SMF and Puppet

Solaris Service Management Facility (SMF) and Puppet represent a synergistic approach to building and maintaining audit-ready systems. SMF is a core component of the Solaris OS, providing deterministic service lifecycle management, dependency handling, and fault recovery. Puppet, by contrast, is a cross-platform configuration management tool that allows

for declarative provisioning, state enforcement, and drift correction. When combined, these tools enable infrastructure that is not only compliant and self-documenting but also resilient to misconfiguration and unauthorized change.

### 3. Motivation for Integrating SMF and Puppet

As organizations move toward automation and governance-by-design, the need for tools that align technical state with policy mandates is critical. Integrating SMF and Puppet allows for seamless orchestration of services and configurations with verifiable outputs. This integration supports automated provisioning, system hardening, and compliance enforcement, reducing both the risk of audit failure and the overhead associated with manual validation. The goal is to build infrastructure that inherently complies with standards and can continuously prove it.

## II. OVERVIEW OF SMF IN SOLARIS

### 1. Key Features of SMF

Solaris SMF replaces traditional startup scripts with a structured, object-oriented service management framework. Key features include dependency resolution, restart contracts,

and centralized state tracking. Each service in SMF is defined as a distinct entity with properties, dependencies, and fault behavior. SMF automatically manages the start order based on service interdependencies, improving reliability during system boot and runtime. Its restart contracts ensure that failed services are restarted according to specified policies, minimizing downtime. Administrators gain fine-grained control over service states, transitions, and recovery mechanisms—critical components for audit-focused environments.

## 2. Manifest Design for Auditability

Services in SMF are defined using XML-based manifests. These manifests encapsulate service properties, methods, and dependencies in a standardized, machine-readable format. Version-controlled and stored in Git or similar repositories, these manifests allow teams to track and audit changes over time. During audits, manifests provide concrete evidence of what services are defined, how they are configured, and how they relate to one another. This structure ensures repeatable service behavior across environments and helps prevent configuration drift or undocumented service additions.

## 3. Fault Management Integration

SMF is tightly integrated with Solaris Fault Management Architecture (FMA), allowing it to respond intelligently to underlying system faults. For instance, hardware or driver failures detected by FMA can trigger service state changes in SMF, such as placing a service into maintenance mode. This provides operational visibility into both software and hardware layers and supports proactive fault mitigation. By correlating faults with service behavior, administrators can build resilient systems that comply with service availability and continuity requirements, which are often mandated by regulatory frameworks.

# III. PUPPET AS AN INFRASTRUCTURE AS CODE (IAC) TOOL

## 1. Declarative System Descriptions

Puppet operates on the principle of declaring the desired state of a system, rather than scripting imperative steps. Each node's configuration is defined using manifests written in Puppet's domain-specific language (DSL), which describe resources like users, packages, files, and services. The Puppet agent compares the system's current state against the desired state defined by the manifest, applying changes where needed. This declarative model enhances reproducibility and auditability—ensuring that configurations are predictable, consistent, and aligned with organizational policies.

## 2. Module Design and Role Classification

Puppet's modular architecture promotes reuse and scalability. Modules encapsulate specific configuration tasks—such as

managing SSH, firewall rules, or software installations—while classes and roles abstract node-specific responsibilities. Nodes can be assigned roles like “database server” or “web proxy,” which in turn include the appropriate classes and modules. This model simplifies management across large-scale UNIX deployments and supports the enforcement of standard configurations, reducing variation and risk. Version-controlled modules also provide an audit trail of configuration logic over time.

## 3. Reporting and Compliance Features

Puppet's built-in reporting engine provides visibility into configuration application, resource changes, and enforcement results. Reports are generated during each agent run and can be visualized through the Puppet Enterprise Console or exported to third-party tools like Foreman or ELK. Puppet also integrates with compliance tools such as OpenSCAP to enforce security baselines. These reporting capabilities enable continuous verification of compliance posture, offer a basis for alerting on unauthorized changes, and support audit documentation. The combination of logs, dashboards, and historical change data strengthens IT governance and audit readiness.

# IV. INTEGRATION PATTERNS BETWEEN SMF AND PUPPET

## 1. Service Registration with Puppet

One of the primary integration points between Puppet and SMF is the automated deployment and registration of service manifests. Puppet can be configured to place SMF XML manifest files into the appropriate directories, such as `/lib/svc/manifest/site`, and subsequently execute the `svccfg import` command to register these services with the SMF repository. This removes the need for manual intervention in service setup and ensures that service definitions are always in sync with infrastructure code. By embedding this logic in Puppet modules, organizations can version-control their service architecture and consistently deploy it across multiple environments. This provides not only operational efficiency but also alignment with auditability and reproducibility standards.

## 2. Lifecycle Management and Enforcement

Through resource declarations, Puppet can manage the lifecycle of SMF services by defining whether services should be enabled, disabled, or in maintenance mode. This allows administrators to enforce specific service states declaratively, ensuring compliance with operational policies. For instance, services not required in non-production environments can be automatically disabled, minimizing the attack surface. The result is a consistent and controlled runtime environment that matches the organization's compliance blueprint.

### 3. Error Recovery and Convergence

SMF's built-in self-healing capability, combined with Puppet's idempotent enforcement model, provides a strong foundation for error recovery. If a service enters an unintended state or is manually altered, Puppet will detect the deviation during its regular run and reassert the correct state. Simultaneously, SMF will attempt to restart failed services based on defined contracts. This dual convergence model ensures that both configuration and service availability are continuously aligned with expected behavior—an essential trait for compliance-focused environments.

## V. BUILDING AUDIT-READY IMAGES

### 1. Golden Image Creation with Puppet and SMF

Creating golden images is a cornerstone of repeatable infrastructure. By embedding both Puppet-managed configurations and SMF service definitions, organizations can produce standardized system images that are policy-compliant by design. Puppet is used to configure user accounts, install approved packages, enforce security baselines, and place service manifests, while SMF ensures that services start in a known-good sequence with defined dependencies and privileges. These images serve as templates for all future deployments, dramatically reducing configuration drift and build-time variance across environments.

### 2. Immutable Build Patterns

Audit-ready environments benefit greatly from immutable infrastructure principles, where system instances are never modified post-deployment but instead rebuilt from known, version-controlled templates. Puppet supports this through environment segregation and Git-based code repositories, while SMF manifests are similarly versioned and embedded in the image. By assigning unique identifiers to each image build—such as commit hashes and timestamps—organizations can ensure that each deployed node is traceable back to its source configuration. This traceability is a key requirement in standards such as ISO 27001 and NIST 800-53.

### 3. Audit Trail Embedding in Build Process

Integrating audit metadata directly into the build process further enhances traceability and compliance. Puppet can tag systems with data such as module versions, build numbers, and repository states, while SMF services can log their operational metadata during boot. These elements provide auditors with a verifiable chain of evidence demonstrating that the deployed systems were built and configured according to defined security and operational standards. Combined, they make post-deployment audits faster, more accurate, and easier to automate.

## VI. COMPLIANCE ENFORCEMENT MECHANISMS

### 1. Role of Puppet in Policy Adherence

Puppet plays a critical role in ensuring that systems remain compliant with security and operational policies such as CIS benchmarks or DISA STIG requirements. By expressing these policies as reusable Puppet modules, organizations can codify compliance logic into their infrastructure as code (IaC) framework. These modules can be applied across various environments, enabling consistent enforcement of password policies, file permissions, network configurations, and more. Puppet's idempotent design ensures that non-compliant configurations are automatically corrected at regular intervals, thereby minimizing human error and configuration drift. This model supports continuous compliance without manual intervention, aligning with the needs of audit-bound industries such as banking, defense, and healthcare.

### 2. SMF-Based Security Controls

SMF complements Puppet by allowing fine-grained control over service behaviors through property groups and execution contexts. Administrators can use SMF to restrict service privileges, define resource limits, and enforce dependency constraints. This ensures that services cannot exceed their defined operational scope, reducing the attack surface of Solaris systems. Timeout properties and restart thresholds can also be configured to prevent denial-of-service or hung-state conditions, which are often red flags in compliance audits.

### 3. Continuous Compliance Reporting

Together, Puppet and SMF enable real-time visibility into system compliance. Puppet's reporting engine generates logs and compliance summaries, which can be ingested into SIEM tools. SMF service logs contribute detailed operational context, enhancing audit readiness.

## VII. CONFIGURATION DRIFT AND RECONCILIATION

### 1. Detecting and Correcting Drift with Puppet

Configuration drift occurs when a system's current state deviates from its intended configuration, often due to ad-hoc changes, patching, or malicious activity. Puppet addresses this by continuously comparing the system's actual state against its declared manifests. When drift is detected, Puppet's agent automatically remediates it, ensuring that the system returns to a compliant state without manual intervention. This approach provides not only operational stability but also the consistency necessary for audit-readiness. Scheduled Puppet runs act as periodic self-healing events, maintaining infrastructure integrity and reinforcing policy adherence.

## 2. SMF Alerts and Service State Deviation

SMF provides native mechanisms for identifying and reacting to service state changes. Services can transition to "maintenance" mode due to dependency issues or startup failures, often indicating misconfigurations or environmental anomalies. By integrating SMF alerts into system monitoring pipelines, administrators can be notified in real-time when critical services deviate from their expected states. These alerts help maintain service availability while contributing to audit trail completeness.

## 3. Event Correlation and Forensics

A powerful aspect of combining Puppet and SMF is the ability to correlate configuration changes with service-level impacts. Puppet logs reveal what was changed, while SMF provides context on how those changes affected service health. This correlation enhances incident response and forensic investigation, enabling auditors to trace causality between configuration actions and system behavior—critical in post-breach investigations or compliance inquiries.

# VIII. LOGGING AND AUDIT TRAIL GENERATION

## 1. Centralized Logging of Puppet Activities

For audit-readiness, capturing and aggregating logs from Puppet activities is essential. Puppet master servers produce detailed logs of resource application, changes made, and any deviations from expected states. These logs, including run reports and summaries, can be exported to centralized logging platforms like Graylog, Splunk, or the ELK stack for long-term retention, analysis, and dashboarding. This centralization simplifies compliance auditing, as administrators can produce historical evidence of configuration enforcement on demand.

## 2. SMF Log Integration

SMF maintains its own logs in `/var/svc/log`, which include information about service start-ups, shutdowns, errors, and state transitions. These logs can be ingested alongside Puppet data to provide a full-spectrum view of system state and operations. For example, a service failure log can be correlated with a recent configuration change applied by Puppet, giving security and compliance teams better visibility and context into system behavior.

## 3. Audit Record Structuring for Compliance

Both Puppet and SMF logs can be normalized and structured to align with regulatory standards such as SOC 2, HIPAA, or ISO 27001. Key metadata like timestamps, user attribution, configuration digests, and system identity can be embedded in log entries, enabling automated parsing and report generation. These structured records are critical during audits, allowing reviewers to confirm compliance with change management, access control, and service continuity requirements.

# IX. OPERATIONAL USE CASES

## 1. Healthcare System Build Automation

In healthcare IT, audit-readiness is fundamental due to HIPAA regulations and the sensitivity of patient data. Puppet and SMF together enable automated provisioning of Solaris-based EHR, PACS, and imaging servers with predefined configurations and hardened services. For example, Puppet modules can enforce encrypted communication channels, manage user access controls, and validate software versions, while SMF ensures all clinical services are booted in the correct order and monitored for availability. By integrating Puppet with change management tools and tagging SMF services, hospitals can achieve system lineage traceability. This automation not only reduces setup time but also guarantees that deployed servers meet compliance expectations out of the box.

## 2. Finance Sector: High-Compliance Environments

Financial institutions require strict compliance with PCI-DSS, SOX, and internal governance policies. Puppet facilitates centralized control over configuration baselines and enforces cryptographic settings, logging configurations, and access restrictions. SMF, on the other hand, ensures that services are resilient and governed by strict startup and dependency rules. Together, they deliver robust build pipelines for financial application servers that support real-time auditing, fault recovery, and state enforcement. Service digests and Puppet run reports can be logged to SIEM systems for continuous compliance monitoring, forming a solid defense against configuration-related vulnerabilities.

## 3. Research Clusters with Traceable Provenance

In scientific and academic computing, reproducibility is paramount. Research platforms often require high-performance Solaris servers running custom compute or data pipelines. With Puppet managing environmental consistency and SMF supervising services and job daemons, researchers can ensure that computational environments are identical across nodes and time. Build artifacts, job logs, and SMF states can be stored with experiments, supporting provenance tracking for peer review or regulatory scrutiny. This methodology is particularly valuable in fields like genomics, physics, and climate modeling, where data integrity and traceability are central.

# X. SECURITY ENHANCEMENTS AND HARDENING

## 1. Puppet-Driven Kernel and Package Hardening

Security hardening begins with the operating system. Puppet allows administrators to enforce OS-level configurations such as kernel parameters (e.g., `sysctl` settings), filesystem mount options (e.g., `noexec`, `nodev`), and controlled package

installations. Puppet modules aligned with CIS or DISA STIG standards can be applied across nodes to ensure consistent compliance. These modules are versioned and audited, enabling traceable security hardening across the infrastructure. As new threats emerge, updates to these modules can be propagated automatically, maintaining secure baselines without delay.

### 2. SMF Security Contexts and Privileges

SMF introduces fine-grained control over service execution through context-based privileges and resource constraints. Each service can be configured to run with only the permissions it requires, based on Solaris's RBAC and privilege model. For example, a web server SMF instance may be isolated from kernel-level access or unnecessary file systems. These restrictions limit the potential impact of compromised services and help maintain principle-of-least-privilege adherence. Logging each privilege assignment also aids in audit preparation and post-incident forensics.

### 3. Vulnerability Mitigation via Automated Patching

Patch management is critical in maintaining system integrity. Puppet can schedule patch deployments, verify package versions, and trigger reboots or service restarts when needed. SMF supports seamless restarts of services through defined restart contracts, ensuring that essential services are automatically brought back online after patching. By combining Puppet's enforcement with SMF's reliability, organizations can reduce manual intervention during patch windows, lower downtime, and maintain an auditable history of security updates applied further contributing to regulatory compliance and operational readiness.

### Challenges and Mitigation Strategies

#### Manifest Complexity and Maintenance

Managing large-scale SMF deployments can become complex, especially when dealing with deeply nested service dependencies and custom manifest designs. Over time, manifests can grow difficult to maintain or validate. To mitigate this, organizations can modularize SMF configurations and store them in version control systems alongside Puppet code. Tools such as `svccfg` can be scripted through Puppet to validate manifests during the CI/CD process, reducing human error and improving maintainability.

#### Puppet Master Bottlenecks

In environments with thousands of nodes, the Puppet master server can become a bottleneck due to concurrent catalog compilations and resource consumption. Performance issues may lead to delayed or failed configuration enforcement. Solutions include scaling the Puppet infrastructure horizontally, using compilers and load balancers, or leveraging PuppetDB to offload data query tasks. Additionally, employing cached catalogs on agents can reduce master dependence during transient outages.

### Error Debugging Across Layers

When configuration or service issues arise, diagnosing problems that span Puppet, SMF, and the OS can be time-consuming. For example, a misconfigured manifest might deploy successfully but cause a service to enter maintenance mode in SMF. To streamline troubleshooting, logs from Puppet, SMF (`/var/svc/log`), and the system journal should be integrated into a centralized dashboard (e.g., ELK or Splunk). Cross-referencing logs by timestamp and service name allows administrators to pinpoint root causes quickly and resolve issues with minimal disruption.

### Future Trends in Audit-Ready Automation

#### Policy-as-Code and Compliance-as-Code

The shift toward treating compliance as part of infrastructure design is accelerating through practices like Policy-as-Code (PaC) and Compliance-as-Code (CaC). These paradigms embed compliance requirements directly into the system provisioning workflow using tools such as Puppet, InSpec, and Open Policy Agent (OPA). Rather than documenting compliance separately, organizations codify rules that enforce secure configurations, user access limits, and encryption settings. These policies can be tested, version-controlled, and enforced just like any application logic. Puppet's role in PaC allows teams to manage regulatory mandates declaratively and ensure continuous enforcement across deployments. This leads to auditable, testable infrastructures that evolve with changing compliance requirements while reducing human error and audit preparation time.

#### Integration with AIOps and Observability Platforms

Future-ready infrastructures are increasingly integrating with AIOps platforms and observability tools to automate issue detection, compliance drift alerts, and service health assessments. Puppet reports and SMF service telemetry can be fed into systems like Splunk, Prometheus, or ELK for correlation and anomaly detection. AIOps can analyze logs, suggest remediation actions, and even trigger Puppet runs or SMF service restarts. This convergence of configuration management and observability will enable predictive compliance, self-correcting infrastructure, and smarter decision-making shifting organizations from reactive audit practices to proactive governance.

#### Immutable Infrastructure and Zero Trust Builds

Immutable infrastructure ensures that systems are never modified post-deployment. Instead, they are rebuilt from trusted images with pre-applied configurations and compliance policies. Coupled with Zero Trust principles where no service or user is inherently trusted these builds minimize attack surfaces. Puppet helps maintain consistency in the image build pipeline, while SMF ensures controlled service behavior within these locked-down systems. The adoption of immutability and Zero Trust will become a cornerstone of audit-ready design, particularly as systems

become more ephemeral and containerized in cloud and hybrid environments.

## XI. CONCLUSION

Achieving audit-ready system builds in modern UNIX environments demands more than manual checklists and retroactive compliance assessments—it requires automation, precision, and transparency at every layer. The combination of Solaris SMF and Puppet presents a powerful architectural model for delivering systems that are not only robust and scalable but also inherently verifiable and compliant. SMF provides a deterministic service orchestration framework with built-in fault handling and monitoring, while Puppet delivers declarative provisioning, drift management, and continuous policy enforcement. Together, they support repeatable system builds, enforce security baselines, and maintain detailed logs and service metadata essential for audit purposes. Use cases across healthcare, finance, and scientific research illustrate their effectiveness in delivering infrastructure with high trustworthiness and operational continuity. As infrastructure evolves, emerging trends such as Policy-as-Code, integration with observability tools, and immutable system design will further enhance audit-readiness. By aligning infrastructure management with compliance objectives, organizations can significantly reduce the risks and costs associated with audits while improving overall IT governance. Ultimately, the SMF-Puppet integration fosters a future where audit-readiness is not a periodic project but a continuous operational capability, embedded directly into the fabric of infrastructure lifecycle management.

## REFERENCES

1. Pamungkas, A.T. (2020). Making a "Pandhawa Lima" Puppet Puzzle Game Android based using Unity3D.
2. Huang, A., Huang, F., & Jhu, J. (2018). Unreal Interactive Puppet Game Development Using Leap Motion. *Journal of Physics: Conference Series*, 1004.
3. Collard, J.M., Gupta, N., Shambaugh, R., Weiss, A., & Guha, A. (2015). On Static Verification of Puppet System Configurations. *ArXiv*, abs/1509.05100.
4. Herawaty, E., & Susiloatmadja, R. (2011). BUILDING OF SMF DITKESAD REGISTRATION WEBSITE USING MACROMEDIA DREAMWEAVER 8, PHP AND MYSQL.
5. Hendrix, V.C., Benjamin, D., & Yao, Y. (2012). Scientific Cluster Deployment and Recovery – Using puppet to simplify cluster management. *Journal of Physics: Conference Series*, 396, 042027.
6. Merrill, H.W. (1974). On using statistical analysis of SMF data to build usable simulation models of large-scale computer systems. *Annual Simulation Symposium*.
7. Rett, J., & Dias, J. (2005). Visual Based Human Motion Analysis: Mapping Gestures Using a Puppet Model. *Portuguese Conference on Artificial Intelligence*.
8. Turnbull, J. (2008). Pulling Strings with Puppet : Configuration Management Made Easy.
9. Russell-Yates, R., & Southgate, J. (2018). *Mastering Puppet 5*.
10. Battula, V. (2021). Dynamic resource allocation in Solaris/Linux hybrid environments using real-time monitoring and AI-based load balancing. *International Journal of Engineering Technology Research & Management*, 5(11), 81–89. <https://ijetrm.com>
11. Battula, V. (2022). *Legacy systems, modern solutions: A roadmap for UNIX administrators*. Royal Book Publishers.
12. Madamanchi, S. R. (2021). Disaster recovery planning for hybrid Solaris and Linux infrastructures. *International Journal of Scientific Research & Engineering Trends*, 7(6), 01–08.
13. Madamanchi, S. R. (2021). Linux server monitoring and uptime optimization in healthcare IT: Review of Nagios, Zabbix, and custom scripts. *International Journal of Science, Engineering and Technology*, 9(6), 01–08.
14. Madamanchi, S. R. (2021). *Mastering enterprise Unix/Linux systems: Architecture, automation, and migration for modern IT infrastructures*. Ambisphere Publications.
15. Madamanchi, S. R. (2022). *The rise of AI-first CRM: Salesforce, copilots, and cognitive automation*. PhDians Publishers.
16. Mulpuri, R. (2021). Command-line and scripting approaches to monitor bioinformatics pipelines: A systems administration perspective. *International Journal of Trend in Research and Development*, 8(6), 466–470.
17. Mulpuri, R. (2021). Securing electronic health records: A review of Unix-based server hardening and compliance strategies. *International Journal of Research and Analytical Reviews*, 8(1), 308–315.
18. Krum, S., Hevelingen, W.V., Kero, B., Turnbull, J., & McCune, J. (2013). *Getting Started with Puppet*.
19. Zhou, J., Shah, R., Guo, X., Du, C., & Wang, X. (2020). All-optical fiber ultrasound imaging system based on the photoacoustic principle. *Medical Imaging*.
20. Alateeq, A.S., & Matin, M.A. (2013). Using M-QAM-OFDM in Downstream Link and WBTW-SOA in Upstream of a Bidirectional WDM-PON System. *Optics and Photonics Journal*, 03, 88-92.
21. Arundel, J. (2011). *Puppet 3 Cookbook*.
22. Ramos, E., & Castro, C. (2012). *Kinect Networked Puppet*.
23. Hafiz, M.Z., & Jun, M. (2011). *The Shadow Augmented Reality Puppet*