# 🐻❄️ Puppet – Cheatsheet del Flujo de Trabajo Completo

> **Objetivo:** Entender de punta a punta cómo Puppet impone el estado deseado en los sistemas mediante un ciclo de ejecución idempotente, auditable y seguro.

---

## 🧠 1. Arquitectura Básica de Puppet

- **Puppet Master / Server**
  - Procesa facts.
  - Compila catálogos.
  - Expone módulos y manifests.
- **Puppet Agent**
  - Corre en cada nodo gestionado.
  - Envía facts al master, aplica catálogos, envía reportes.
- **PuppetDB** (opcional)
  - Base de datos para almacenar reports, facts y exportar recursos.

---

# 🔄 2. El Ciclo de Ejecución del Agente (Core Workflow)

El flujo se compone de **cuatro etapas clave**, repetidas de forma periódica:

---

## Etapa 1 — Envío de Facts

El agente ejecuta `facter` y recolecta información del sistema:

- SO, red, hardware
- Servicios, IPs
- Variables personalizadas

Luego envía estos **facts** al Puppet Master.

> 💡 Esto permite que los manifests puedan usar lógica condicional basada en la infraestructura real.

---

## Etapa 2 — Compilación del Catálogo

El Puppet Master:

1. Recibe los facts del nodo.
2. Lee los manifests y módulos.
3. Evalúa:
    - Clases incluidas
    - Templates
    - Jerarquía Hiera
    - Parámetros
4. **Compila un Catálogo final**, estático y específico para ese nodo.

> 📄 **El catálogo es una lista ordenada de recursos + sus dependencias**, lista para aplicar.

---

## Etapa 3 — Aplicación del Catálogo (Convergencia)

El agente:

1. Descarga el catálogo.
2. Evalúa cada recurso en orden.
3. Cambia solo lo necesario para pasar del **estado actual** → **estado deseado**.
4. Respeta relaciones:
    - `require`
    - `before`
    - `notify`
    - `subscribe`

> ⚙️ Puppet **no ejecuta comandos arbitrarios**: solo aplica cambios para alcanzar el estado especificado.

---

## Etapa 4 — Envío del Informe (Reporting)

Una vez aplicado el catálogo:

- El agente genera un informe detallado.
- Lo envía al master o a PuppetDB.

El informe indica:

- Qué recursos cambiaron
- Qué recursos estaban ya en estado correcto
- Fallos (si hubo)

> 📊 Esto permite auditoría, monitoreo y troubleshooting.

---

# 🌟 3. Por Qué Este Flujo Asegura Resultados Confiables

✔ Idempotencia

Correr Puppet 10 veces produce siempre el mismo resultado final: el estado deseado.

✔ Auditoría

Los informes permiten rastrear todos los cambios.

✔ Seguridad

El sistema valida, compila y aplica catálogos en capas controladas.

---

# 🛠️ 4. El Ciclo de Vida del Cambio en Puppet

Además del ciclo del agente, Puppet promueve un workflow seguro para aplicar cambios.

---

## Validación Previa

Antes de poner cambios en producción:

- **Validar sintaxis**

```
puppet parser validate manifest.pp
```

- **Simular cambios (sin aplicar)**

```
puppet agent --test --noop
```

> 🔍 Esto permite detectar errores sin modificar nada.

---

## Despliegue Controlado

Puppet recomienda siempre:

### 🔷 Environments

Ejemplo:

- `production`
- `staging`

- `dev`

Cada uno con su propia carpeta de código.

### 🔹 Deploy por fases

Aplicar cambios de forma escalonada:

1. Un nodo de prueba
2. Un subconjunto de la flota
3. Todo el entorno

### 🔹 Code review y pipelines

Repositorios Git + R10K + control de versiones.

---

## 📦 5. El Rol de los Recursos y los Módulos

### Recursos

Son la unidad mínima: `file`, `package`, `service`, `user`, etc.

### Módulos

Carpetas estructuradas que agrupan:

- Manifests
- Templates
- Archivos
- Clases

Puppet compila catálogos usando estos módulos y classes.

---

## ⏱️ 6. Frecuencia del Ciclo

Por defecto:

```
Cada 30 minutos
```

(ajustable con `runinterval`)

---

## 🧭 7. Resumen Visual del Flujo

```
AGENTE → Envía facts
        ↓
MASTER → Compila catálogo
        ↓
AGENTE → Aplica catálogo
        ↓
AGENTE → Envía reporte
```

Y el ciclo vuelve a empezar.