🚀 QUICK START - blinkchamber v2.0

⚡ Inicio Ultra-Rápido (1 Comando)

```
# ✨ Magic command - ¡Esto es todo lo que necesitas!
./scripts/vault-bootstrap.sh all
```

⏱️ **Tiempo**: ~5-7 minutos
🖥️ **Requisitos**: 8GB RAM, 4 CPU cores
✅ **Resultado**: Sistema completo con Vault funcionando

🎯 ¿Qué Obtienes?

Después de ejecutar el comando anterior, tendrás:

- 🔐 **Vault**: Gestión centralizada de secretos
- 🌐 **Zitadel**: Sistema de identidad y autenticación
- 🗄️ **PostgreSQL**: Base de datos segura
- 📊 **Grafana**: Monitoreo y dashboards
- 💾 **MinIO**: Almacenamiento de objetos
- 🔒 **Certificados TLS**: Automáticos con cert-manager

📋 Pre-requisitos (Instalación de 2 minutos)

Opción 1: Script Automático (Recomendado)

```
# Instala todo automáticamente
curl -s
https://raw.githubusercontent.com/blinkchamber/blinkchamber/main/scripts/in
stall-deps.sh | bash
```

Opción 2: Instalación Manual

```
# Kind (Kubernetes local)
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64
chmod +x ./kind && sudo mv ./kind /usr/local/bin/kind

# kubectl
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x kubectl && sudo mv kubectl /usr/local/bin/kubectl

# Terraform
sudo apt-get update && sudo apt-get install -y gnupg software-properties-
```

```
common
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/hashicorp-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list
sudo apt update && sudo apt install terraform

# Helm
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 |
bash

# jq
sudo apt-get install -y jq
```

🚀 Métodos de Despliegue

⚡ Método 1: Todo Automático (Recomendado)

```
# 1. Crear cluster
kind create cluster --name blinkchamber --config config/kind-config.yaml

# 2. Bootstrap completo
./scripts/vault-bootstrap.sh all

# 3. ¡Ya está! Verifica el estado
./scripts/vault-bootstrap.sh status
```

🔧 Método 2: Paso a Paso (Para debugging)

```
# 1. Crear cluster
kind create cluster --name blinkchamber

# 2. Bootstrap por fases
./scripts/vault-bootstrap.sh 1    # Infraestructura básica (2 min)
./scripts/vault-bootstrap.sh 2    # Inicialización Vault (1 min)
./scripts/vault-bootstrap.sh 3    # Configuración secretos (30s)
./scripts/vault-bootstrap.sh 4    # Aplicaciones (2 min)

# 3. Verificar
./scripts/vault-bootstrap.sh status
```

🧪 Método 3: Testing First (Robusto v2.1) ⭐

```
# 1. Test robusto sin conflictos de puertos
./scripts/test-robust-framework.sh parallel \
```

```
    "scenarios:test_all_scenarios:" \
    "integration:test_integration:"

# 2. Si todo está OK, desplegar en serio
./scripts/vault-bootstrap.sh all

# 3. Limpieza automática garantizada
./scripts/test-robust-framework.sh cleanup
```

🔬 Método 4: Testing Específico (Robusto v2.1) ⭐

```
# 1. Test individual completamente aislado
./scripts/test-robust-framework.sh isolated scenario_dev test_dev_complete
minimal

# 2. Tests paralelos seguros sin conflictos
./scripts/test-robust-framework.sh parallel \
    "dev:test_dev_scenario:complete" \
    "staging:test_staging_scenario:minimal"

# 3. Demo de mejoras implementadas
./scripts/test-demo-improvements.sh comparison
```

🧪 Método 3 (Clásico): Testing First

⚠ **Nota**: El método clásico puede tener conflictos de puertos. Se recomienda usar el Método 3 Robusto arriba.

```
# 1. Test comprehensivo (crea cluster temporal)
./scripts/test-master.sh comprehensive  # ⚠ Posibles conflictos de puertos

# 2. Si todo está OK, desplegar en serio
./scripts/vault-bootstrap.sh all

# 3. Limpiar recursos de test
./scripts/test-vault-bootstrap.sh cleanup
```

🔬 Método 4 (Clásico): Testing Específico

```
# 1. Test de escenario específico
./scripts/test-scenarios.sh --scenario dev-complete-tls

# 2. Test de fase específica
./scripts/test-phases.sh --phase 2 --environment staging
```

```
# 3. Test de integración
./scripts/test-integration.sh --environment production
```

🌐 Acceso Inmediato a Servicios

🎯 URLs Principales

Servicio	Comando de Acceso	URL
Vault UI	`./scripts/vault-bootstrap.sh port-forward`	http://localhost:8200/ui
Zitadel	Navegador + DNS local	https://zitadel.blinkchamber.local
Grafana	Navegador + DNS local	https://grafana.blinkchamber.local
MinIO	Navegador + DNS local	https://minio.blinkchamber.local

🔧 Configuración DNS Local (30 segundos)

```
# Agregar dominios a /etc/hosts
sudo tee -a /etc/hosts << EOF
127.0.0.1 vault.blinkchamber.local
127.0.0.1 zitadel.blinkchamber.local
127.0.0.1 grafana.blinkchamber.local
127.0.0.1 minio.blinkchamber.local
EOF
```

🔐 Acceso a Vault

```
# Port-forward automático
./scripts/vault-bootstrap.sh port-forward

# En otra terminal, configurar Vault CLI
source data/vault/vault-env.sh

# Ver secretos
vault kv list secret/
vault kv get secret/database/postgres
```

🏷️ Comandos Esenciales

🚀 Despliegue

Comando	Descripción	Tiempo
`./scripts/vault-bootstrap.sh all`	Bootstrap completo	~5-7 min
`./scripts/vault-bootstrap.sh 1`	Solo infraestructura	~2 min

Comando	Descripción	Tiempo
`./scripts/vault-bootstrap.sh 2`	Solo Vault init	~1 min

📊 Estado y Monitoreo

Comando	Descripción
`./scripts/vault-bootstrap.sh status`	Estado completo del sistema
`./scripts/vault-bootstrap.sh logs`	Logs de Vault
`./scripts/vault-bootstrap.sh port-forward`	Acceso local a Vault

🔐 Gestión de Vault

Comando	Descripción
`./scripts/vault-bootstrap.sh unseal`	Unseal manual (desarrollo)
`source data/vault/vault-env.sh`	Configurar Vault CLI
`vault status`	Estado de Vault

🧪 Testing Comprehensivo

🛡️ RECOMENDADO: Framework Robusto v2.1 ⭐

Comando	Descripción
`./scripts/test-robust-framework.sh isolated <test> <func> [args]`	Test individual aislado (sin conflictos)
`./scripts/test-robust-framework.sh parallel "test1:func1" "test2:func2"`	Tests paralelos seguros
`./scripts/test-robust-framework.sh cleanup`	Limpieza robusta garantizada
`./scripts/test-robust-framework.sh status`	Estado del framework
`./scripts/test-demo-improvements.sh comparison`	Demo antes vs después
`./scripts/test-demo-improvements.sh single`	Demo test individual
`./scripts/test-demo-improvements.sh parallel`	Demo tests paralelos

🔧 Framework Clásico (Problemas Conocidos)

⚠️ **Nota**: Los comandos clásicos pueden tener conflictos de puertos en ejecución paralela.

Comando	Descripción	Estado
`./scripts/test-master.sh quick`	Test rápido (2 min)	✅ Estable

Comando	Descripción	Estado
`./scripts/test-master.sh comprehensive`	Framework completo con test matrix	⚠ Conflictos paralelos
`./scripts/test-comprehensive.sh`	Test completo con todas las combinaciones	⚠ Conflictos paralelos
`./scripts/test-scenarios.sh`	Tests de escenarios específicos	⚠ Conflictos paralelos
`./scripts/test-phases.sh`	Tests por fases individuales	⚠ Conflictos paralelos
`./scripts/test-integration.sh`	Tests de integración end-to-end	✅ Estable
`./scripts/test-vault-bootstrap.sh full`	Test completo básico (5 min)	✅ Estable
`./scripts/test-vault-bootstrap.sh cleanup`	Limpiar tests	✅ Estable

🎯 Casos de Uso Rápidos

🔥 Demo Rápido (5 minutos)

```
# 1. Todo en un comando
./scripts/vault-bootstrap.sh all

# 2. Verificar que funciona
./scripts/vault-bootstrap.sh status

# 3. Acceder a Vault
./scripts/vault-bootstrap.sh port-forward
# Abrir: http://localhost:8200/ui
```

🧪 Testing antes de Desarrollo (Robusto v2.1) ⭐

```
# 1. Test rápido sin conflictos para verificar entorno
./scripts/test-robust-framework.sh isolated quick_check
test_environment_check

# 2. Si pasa, desplegar para desarrollo
./scripts/vault-bootstrap.sh all

# 3. Limpieza automática garantizada
./scripts/test-robust-framework.sh cleanup
```

🔬 Testing Comprehensivo (Robusto v2.1) ⭐

```
# 1. Test completo paralelo sin conflictos
./scripts/test-robust-framework.sh parallel \
    "scenarios:test_all_scenarios:" \
    "phases:test_all_phases:" \
    "integration:test_integration:"

# 2. Ver estado y reportes
./scripts/test-robust-framework.sh status
firefox test-results/*/final-report.html

# 3. Demo de mejoras vs framework anterior
./scripts/test-demo-improvements.sh comparison
```

🚦 Testing antes de Desarrollo (Clásico)

⚠ **Nota**: Puede tener conflictos de puertos. Se recomienda usar la versión Robusta arriba.

```
# 1. Test rápido para verificar entorno
./scripts/test-master.sh quick  # ⚠ Posibles conflictos en paralelo

# 2. Si pasa, desplegar para desarrollo
./scripts/vault-bootstrap.sh all

# 3. Limpiar test
./scripts/test-vault-bootstrap.sh cleanup
```

🔬 Testing Comprehensivo (Clásico)

```
# 1. Test completo con todas las combinaciones
./scripts/test-master.sh comprehensive  # ⚠ Conflictos en paralelo

# 2. Ver reporte detallado
firefox test-reports/comprehensive-report.html

# 3. Test específico si algo falla
./scripts/test-scenarios.sh --scenario dev-complete-tls
```

🔧 Desarrollo Iterativo

```
# 1. Desplegar fase por fase
./scripts/vault-bootstrap.sh 1    # Base
./scripts/vault-bootstrap.sh 2    # Vault
# ... trabajar en tu código ...
./scripts/vault-bootstrap.sh 3    # Secretos
./scripts/vault-bootstrap.sh 4    # Apps
```

🔐 Credenciales por Defecto

🎯 Acceso a Vault

```
# Token root está en:
source data/vault/vault-env.sh
echo $VAULT_TOKEN
```

🌐 Credenciales de Servicios

Todos los passwords se generan automáticamente y se almacenan en Vault:

```
# Ver credenciales de Grafana
vault kv get secret/monitoring/grafana

# Ver credenciales de MinIO
vault kv get secret/storage/minio

# Ver credenciales de Zitadel
vault kv get secret/identity/zitadel
```

🌍 Configuración por Entorno

💻 Development (Local)

```
# Configuración por defecto
./scripts/vault-bootstrap.sh all
```

- Auto-unseal: Deshabilitado
- Backup: Deshabilitado
- HA: Deshabilitado

🚦 Staging

```
# Con auto-unseal de transit
ENVIRONMENT=staging ./scripts/vault-bootstrap.sh all
```

- Auto-unseal: Transit engine
- Backup: Habilitado
- HA: Deshabilitado

🏭 Production

```
# Con auto-unseal de AWS KMS
ENVIRONMENT=production ./scripts/vault-bootstrap.sh all --auto-unseal
awskms
```

- Auto-unseal: AWS KMS
- Backup: Habilitado
- HA: Habilitado

🛠️ Troubleshooting Rápido

❌ Problemas Comunes

Error: "Cluster no encontrado"

```
kind create cluster --name blinkchamber
```

Error: "Vault sealed"

```
./scripts/vault-bootstrap.sh unseal
```

Error: "No se puede acceder a servicios"

```
# Configurar DNS local
sudo tee -a /etc/hosts << EOF
127.0.0.1 zitadel.blinkchamber.local
127.0.0.1 grafana.blinkchamber.local
EOF
```

🔍 Debugging

```
# Ver estado detallado
./scripts/vault-bootstrap.sh status

# Ver logs de Vault
kubectl logs -f deployment/vault -n vault

# Verificar conectividad
curl -s http://localhost:8200/v1/sys/health | jq
```

🧹 Limpieza

🗑️ Limpieza Completa

```
# Opción 1: Eliminar cluster completo
kind delete cluster --name blinkchamber

# Opción 2: Reset desde dentro
kubectl delete namespace vault identity database storage monitoring --
ignore-not-found
```

🔦 Limpieza de Tests

```
# Limpiar recursos de test
./scripts/test-vault-bootstrap.sh cleanup
```

📈 Siguiente Nivel

Una vez que tengas el sistema funcionando:

1. 📖 **Lee la documentación completa**: README-VAULT-BOOTSTRAP.md
2. 🏗️ **Explora la infraestructura**: terraform/README.md
3. 🔐 **Aprende sobre Vault**: https://learn.hashicorp.com/vault
4. 🌐 **Configura Zitadel**: https://zitadel.com/docs

🆘 Ayuda y Soporte

- 🐛 **Issues**: GitHub Issues
- 💬 **Discussions**: GitHub Discussions
- 📚 **Documentación**: docs/

> 💡 **Pro Tip**: El comando `./scripts/vault-bootstrap.sh all` resuelve el 95% de los casos de uso. ¡Úsalo y luego explora!