

# IT 자동화의 글로벌 동향과 방향

(IT Automation & RPA)

# ANSIBLE은 누구에게 필요한 도구 일까요?



- IT 운영 부담을 줄이려고 하는 개발팀, 시스템 운영팀, 네트워크 운영팀, DEVOPS 팀
  - 아직 Ansible을 사용하고 있지 않으세요?
  - IT 관련 신청 및 접수가 많아서 힘들어요 ?
  - IT 운영이 특정 인력에 의존적이신가요?
  - 애플리케이션 배포와 인프라 변경을 자주하시나요?
  - 운영에 대한 커뮤니케이션 도구가 필요해요?

# 주 52 시간 근무와 IT Automation

주 52 시간 근무로 “워라밸” 실현 가능할까?



줄어드는 근무시간에 대한  
근본적 해결 위한

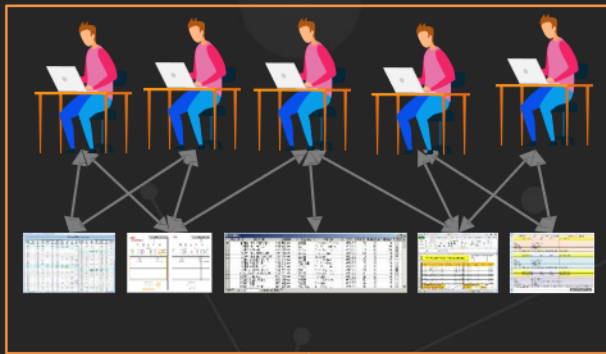
업무 생산성  
강화 방안은?

주 52시간 시대에 효과적으로 대응하기 위해서는  
근무시간 단축 뿐만 아니라 생산성 향상을  
IT 자동화 노력이 필수적임

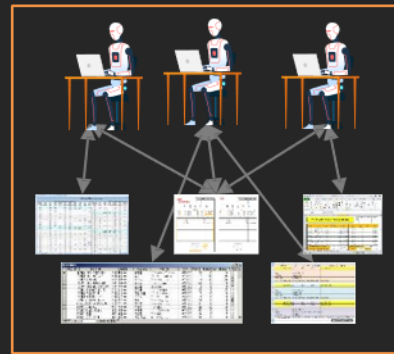
# RPA의 정의

AI 와 머신 러닝 등을 포함한 인지 기술을 활용한 업무 자동화에 대한 노력  
RPA = Robotic Process Automation

[단순 반복적인 업무. Human Resource]



[단순 반복적인 업무. Digital Workforce][고부가가치 업무. Human Resource]

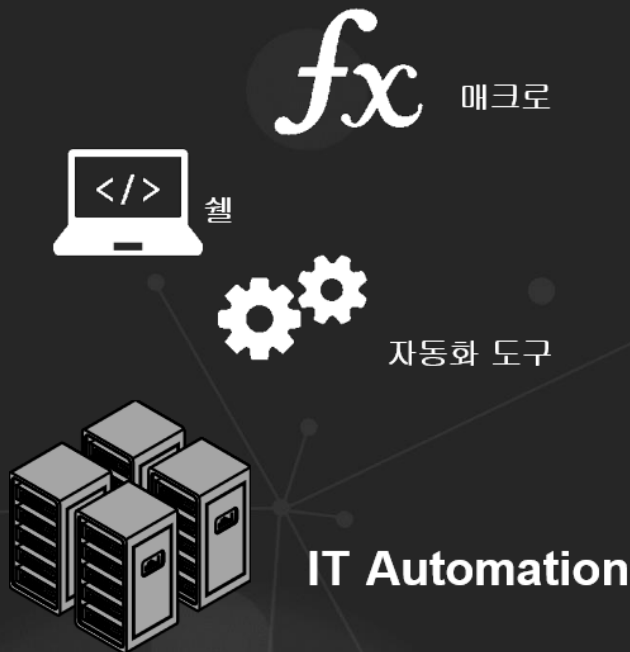


➡  
"RPA 도입"

+



# 업무 생산성을 위한 자동화



- IT 인프라 관리를 자동화
- 정형화된 시스템 작업 위주



- 비즈니스 애플리케이션 사용자를 위한 업무 자동화
- 윈도우 애플리케이션, 엑셀, 웹브라우저를 통한 업무 시스템

# 업무 프로세스 혁신



## Intelligent Enterprise

- 기업 전 영역에 걸쳐 AI/ML, IOT, Analytics등 첨단 기술을 활용하여 고성능을 창출하고 심도 깊은 Insight 제공
- 사람은 보다 고가치 업무에 집중



## AI/ML

- 지능과 판단이 필요한 업무를 머신이 수행 가능
- 데이터가 축적되고 학습이 진행될수록 더 똑똑해짐



## BPR

- 시스템화 + 업무 프로세스 개선
- 프로세스 단위의 자동화 실현



## Automation

- 업무 자동화 혹은 시스템화
- 반복적이고 쉬운 단위 업무 중심



## Digital Transformation

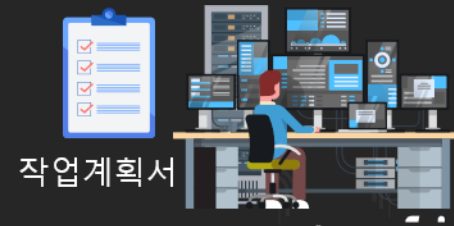
Application Performance Management

Cloud Ready System

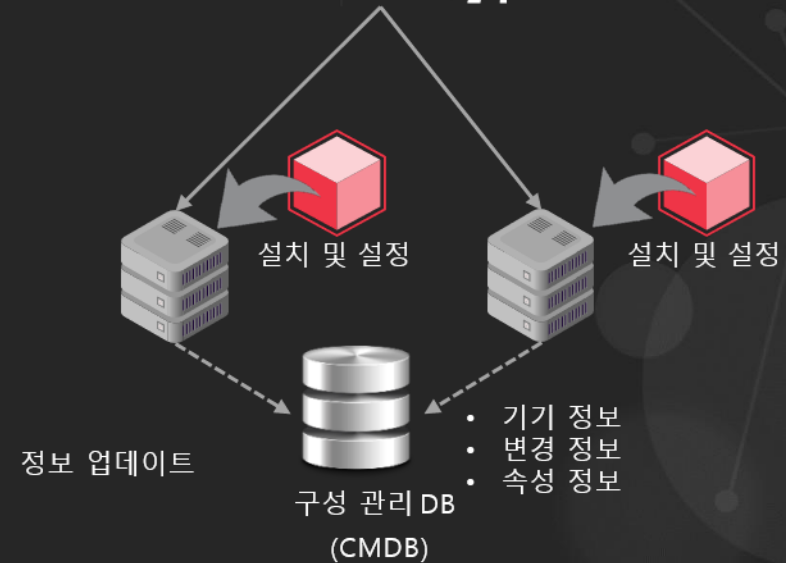


# 기존의 구성 관리 이슈

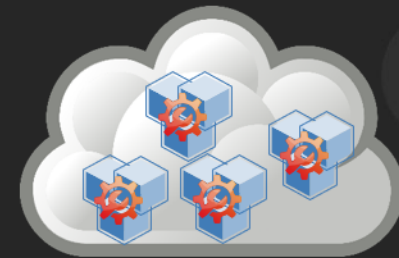
- 기존 온-프레미스 환경에서
  - 엔지니어가 복잡한 인프라 구축
  - 구축 후 만료될 때까지 장기적 운영
  - 미들웨어 클러스터 구축, 수백 대의 OS 초기 설치 및 설정
  - 운영 시 변경 관리와 CMDB 수동 업데이트 등



가상화를 통해 물리적 제약이 없어졌지만 관리 복잡성 증가



- 클라우드 환경으로 전환하면서
  - 인프라 자원의 라이프 사이클이 짧아짐
  - 구축과 운영 비용이 급증

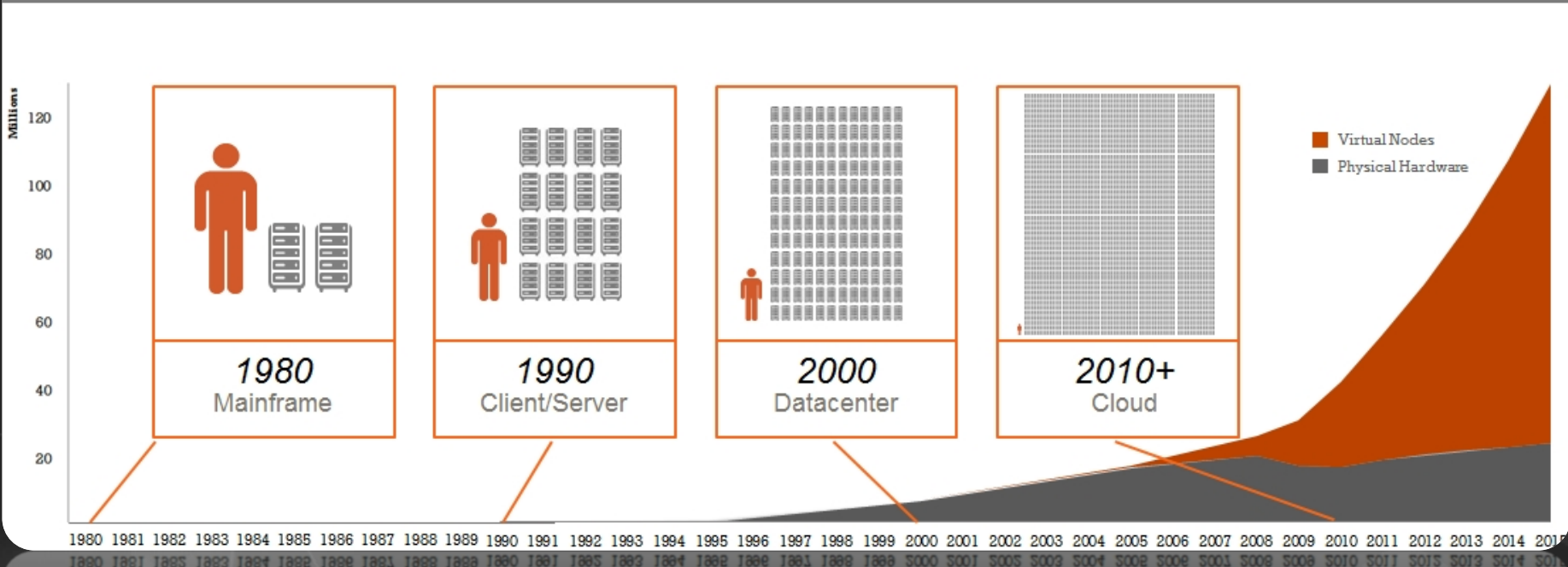




# Increasing scale and complexity means we need admin automation



## Scale x Complexity > Skills



Opscode gets more venture dough for its Chef

From - <http://goo.gl/dLcjS>

# 왜 운용 자동화 시스템이 필요한가요?

장애의 65 %는 **Human Error**이며, 시스템 복잡도와 난이도 증가

시스템 운용 업무의 45 %는 정기적으로 수행해야 하는 반복 작업

운영 효율화를 통한 비용 절감의 요구



시스템의 대규모화



높은 수준의 엔지니어 부족



지속적인 시스템 통합 요구



동일한 작업 반복



운영 품질 향상



운영 비용 (TCO) 절감 요구

업무 확대와 관련 데이터양의 비약적인 증가

가상화, 클라우드 등 다양한 운영 환경의 증가와 관리 효율화 요구

운영 품질에 대한 지속적인 향상 요청

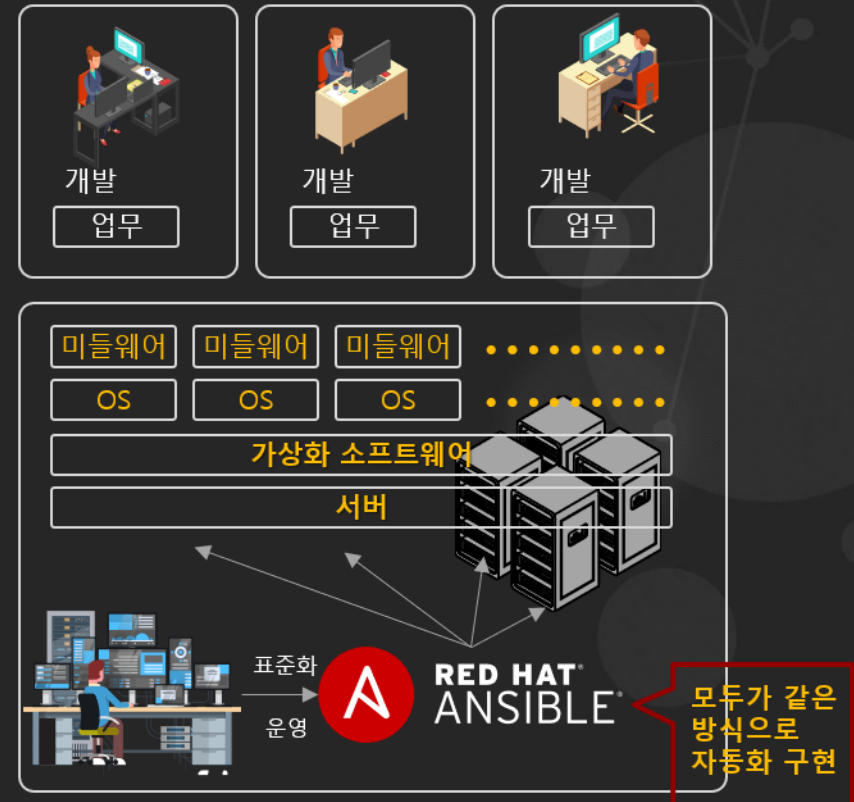
# Ansible 을 이용한 IAC 구현

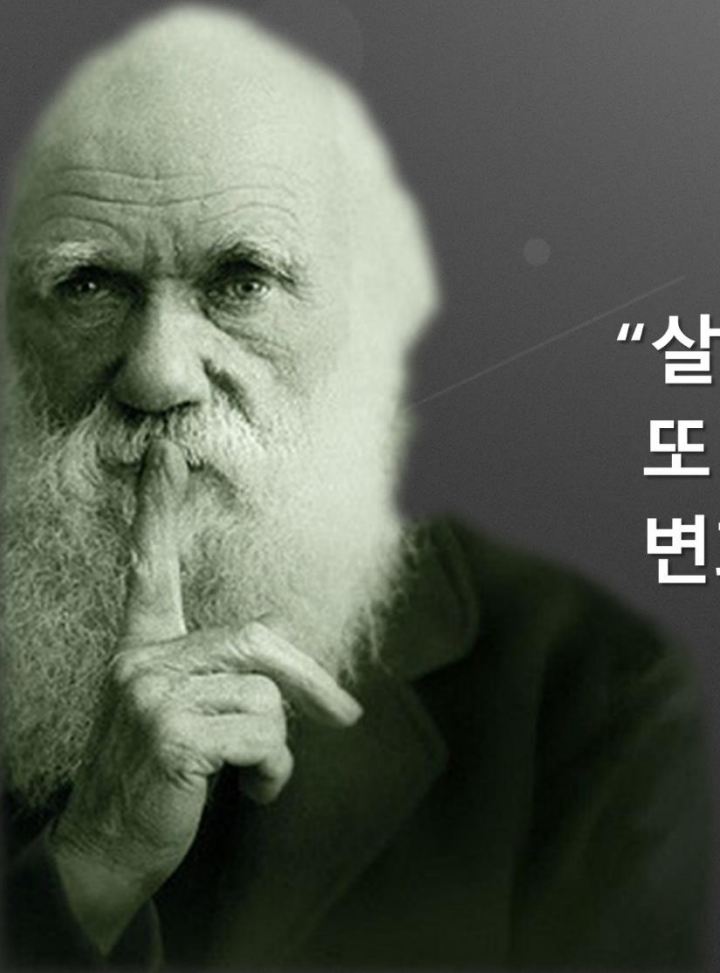
- 시스템 마다 절차와 관리도구가 다르고 수작업에 의한 운영
- Infrastructure As Code 를 통한 시스템 운영 환경의 변화

## 업무 별로 독립적인 운영 환경



## 표준화와 자동화된 운영환경



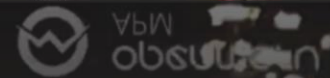


“살아 남는 종(種)은 강한 종이 아니고,  
또 우수한 종도 아니다.  
변화에 적응하는 종이다.”

- *Charles Darwin, 1809*

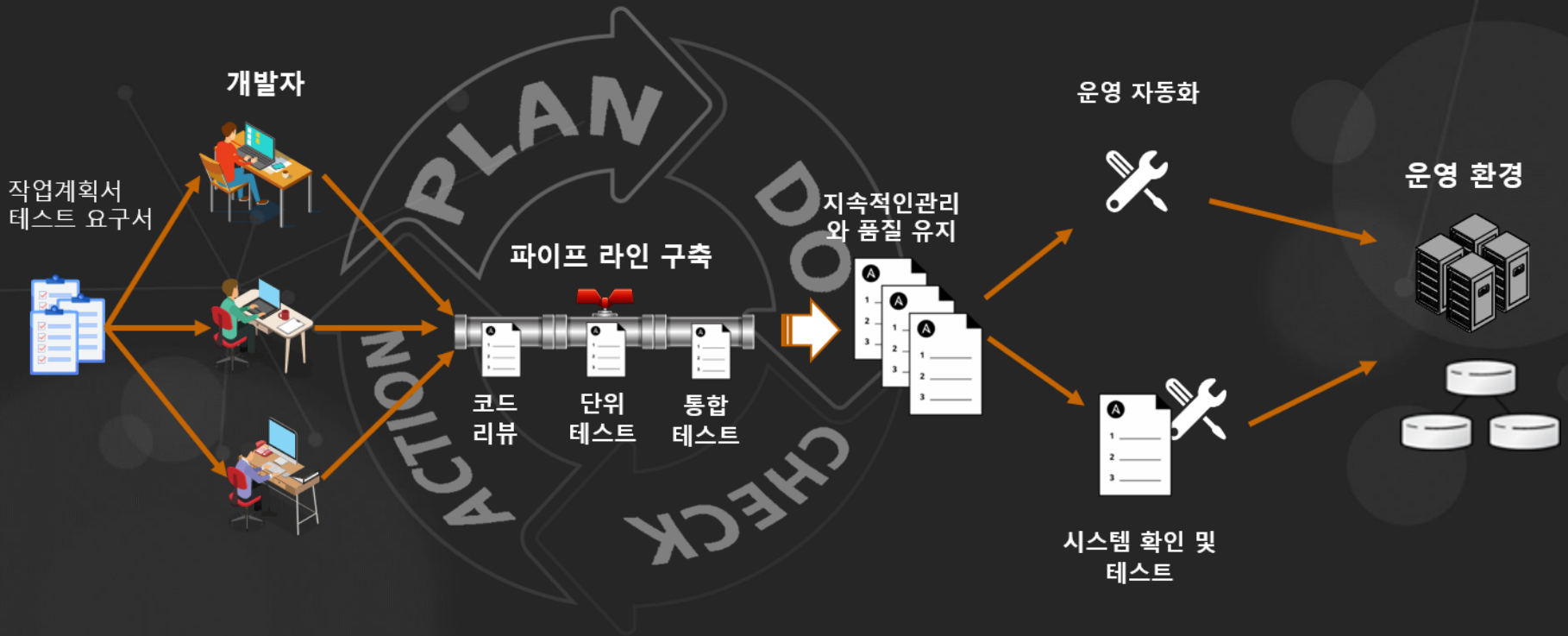
# Application Performance Management

## IAC (Infrastructure As Code)



# Infrastructure as Code

- 수작업으로 했던 인프라 구축 및 변경 작업을 코드로 정의하고 자동화
- 소프트웨어 개발방법론에 따라 개발 과정을 인프라 시스템, 애플리케이션, 미들웨어 배포 및 구성 관리에 적용



# IAC (Infrastructure as Code) – 소프트웨어 개발방법

- IAC = 코딩을 통한 자동화



코드를 통한 장점

자동화 장점

# Infrastructure as Code 혜택

- Infrastructure as Code 을 적용하여 민첩성 높은 서비스를 제공



## 작업 공수 절감

기존 수동으로 해왔던 작업을 코딩 자동화하여 작업 공정 및 납기 단축



## 운영 품질 향상

작업을 코딩하고 자동화하여 작업 품질을 균일하게 유지



## 시스템 운영의 표준화 촉진

- 자동화 및 버전 관리 함으로써 시스템 운영 정책 및 업무 표준화
- 코드를 재사용함으로써 낭비를 제거하고 지속적인 통합 및 배포를 실현



## 작업 통제 강화

작업 작업을 자동화함으로써 내부 통제 및 보안 측면에서의 효과를 기대

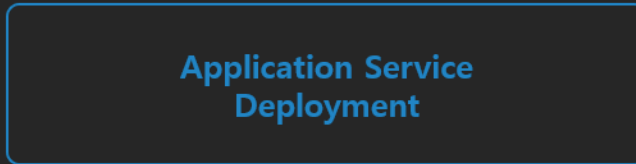


# Infrastructure as Code 적용



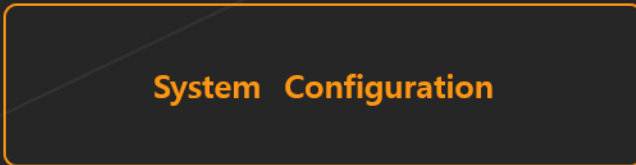
- 각 도구는 Provisioning Toolchain 에 의해 분류

Orchestration  
애플리케이션 배포



Capistrano, Fabric, Serf, Consul etc.

Configuration  
Management  
시스템 구성 관리



Puppet, Chef, Ansible , Itamae

Bootstrapping  
부트스트랩



AWS, Cobbler, Kickstart VMware, Docker



source : Provisioning Toolchain (Velocity2010) by Lee Thompson

# Infrastructure as Code 분류



	설명	작업 예시
Orchestration 애플리케이션 배포	<ul style="list-style-type: none"> <li>• 자원과 관련된 서비스 제공</li> <li>• <b>구축 끝난 환경에서 필요한 작업</b></li> </ul>	<ul style="list-style-type: none"> <li>• <b>애플리케이션 분산 배치</b> (롤링 업데이트)</li> <li>• Hadoop 클러스터 구축</li> <li>• 서비스 신규 추가나 다운을 모니터링</li> <li>• 부하 상황에 따른 Scale Out</li> </ul>
Configuration 시스템 구성 관리	<ul style="list-style-type: none"> <li>• 서버에 <b>미들웨어 설치나 각종 설정</b></li> <li>• <b>서버 구성 관리</b>의 자동화에 해당되는 부분</li> </ul>	<ul style="list-style-type: none"> <li>• OS 설정 ( 유저, 그룹, 서비스와 데몬 ..)</li> <li>• 각종 <b>미들웨어 서버의 구축</b></li> <li>• <b>패치 적용</b></li> </ul>
Bootstrapping 초기 설치	<ul style="list-style-type: none"> <li>• 플랫폼 자체 또는 서버로서 <b>OS가 이용 가능한 상태를 만드는 도구</b></li> <li>• 가상머신이나 컨테이너, 클라우드에 대해 API를 이용해 <b>자원 할당</b></li> </ul>	<ul style="list-style-type: none"> <li>• AWS API 를 이용한 <b>EC2 인스턴스 관리</b></li> <li>• Docker 을 이용한 컨테이너 관리</li> <li>• Vagrant 에서의 가상 머신 생성</li> </ul>

# Instructure As Code 를 위한 Ansible

- Infrastructure as Code 의 모든 영역을 지원



**Orchestration**  
애플리케이션 배포

## Application Service Deployment



**Configuration Management**  
시스템 구성 관리

## System Configuration



**Bootstrapping**  
부트 스트랩

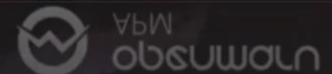
## Cloud or VM - Image Launch OS Install



상표 : 해당 로고는 각 회사 / 단체의 등록 상표입니다.

Application Performance Management

# Ansible 소개



# WHAT IS ANSIBLE AUTOMATION?

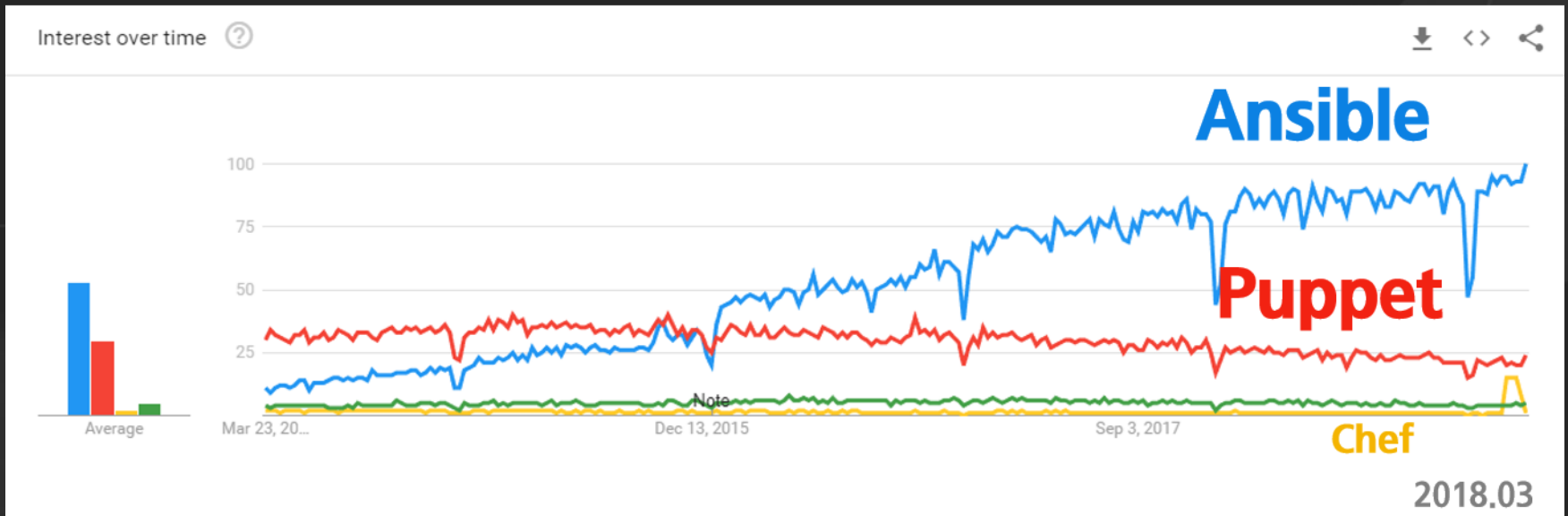
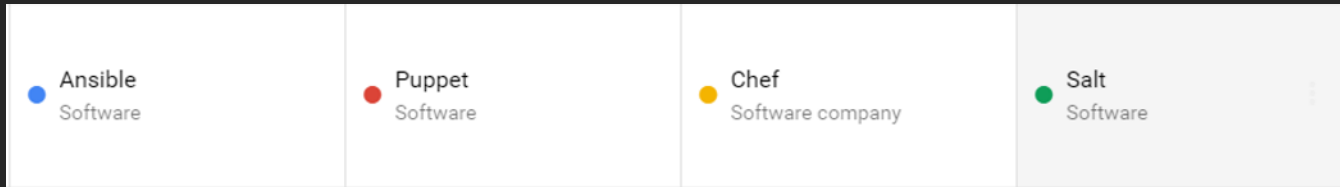


Red Hat acquired Ansible at year end 2015

- *Ansible*은 클라우드 프로비저닝, 구성 관리, 응용 프로그램 배포, 서비스 내
  - 오케스트레이션 및 기타 많은 IT 요구 사항을 자동화하는
  - 간단한 **IT 자동화 엔진**입니다.

“원격 프로토콜을 이용해서 배포, 구성 및 오케스트레이션을 제공하는 자동화 도구”

# Global Google Trends : Ansible vs. Puppet vs. Chef



# Ansible 비교



제품	Puppet	Chef	Ansible
벤더	Puppet Labs	Opscode	Red Hat ( 이전 Ansible.Inc)
출시	2005 년	2009 년	2012 년
구현 언어	Ruby	Ruby/Erlang (서버)	Python
라이선스	Apache License Ver2.0	Apache License Ver2.0	GNU Public License Ver3
도입 사례	○	◎	◎
정보량	○	◎	◎
확장성	◎	◎	◎
Web UI	◎	◎	◎ (Ansible Tower)
정의 파일	자체 DSL (Ruby 기반)	YAML	YAML
코드 관리	manifest	recipe	playbook
에이전트 설치	Pull (Agent)	Pull (Agent)	Push ( Agentless )
시스템 복잡성	△	△	◎

# ANSIBLE FEATURES



## Agentless

- 에이전트가 필요 없는 환경
- OpenSSH & WinRM 지원
- 즉각적인 사용 가능
- 높은 효율성과 보안성

Architecture



## Simple

- YAML 형식의 읽고 쓰기 쉬운 설정 파일
- 프로그래밍 스킬이 필요하지 않음
- 팀 간의 작업 공유가 쉬움
- 멱등성 지원
- 높은 생산성

Playbook



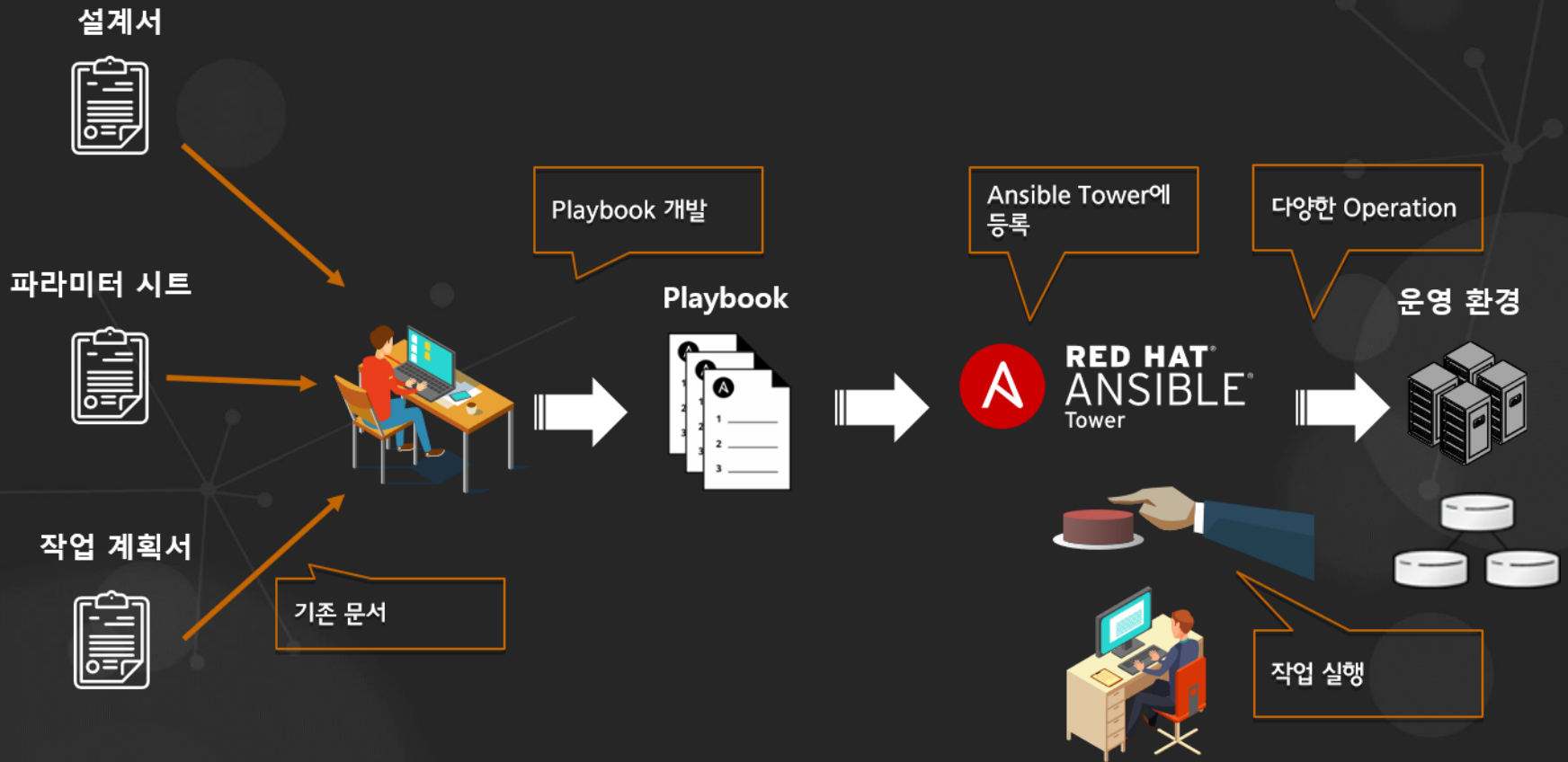
## Powerful

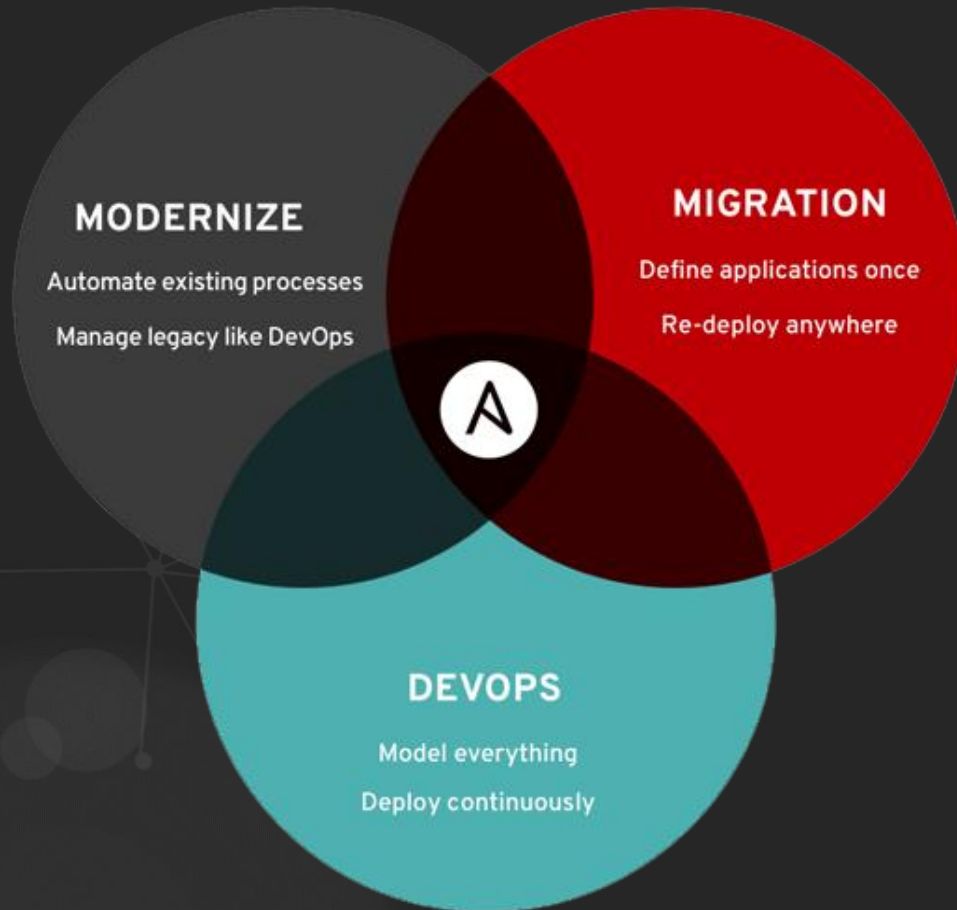
- 700개 이상 대다수의 서버와 네트워크 장비 지원
- 동시에 다수의 대상 서버에서 실행
- Bootstrap 부터 설정 변경까지 원스톱 실행
- 완벽한 구성 관리, 오케스트레이션, 배포

Modules



# Ansible 에서의 자동화의 흐름





# Application Performance Management

# Ansible Architecture



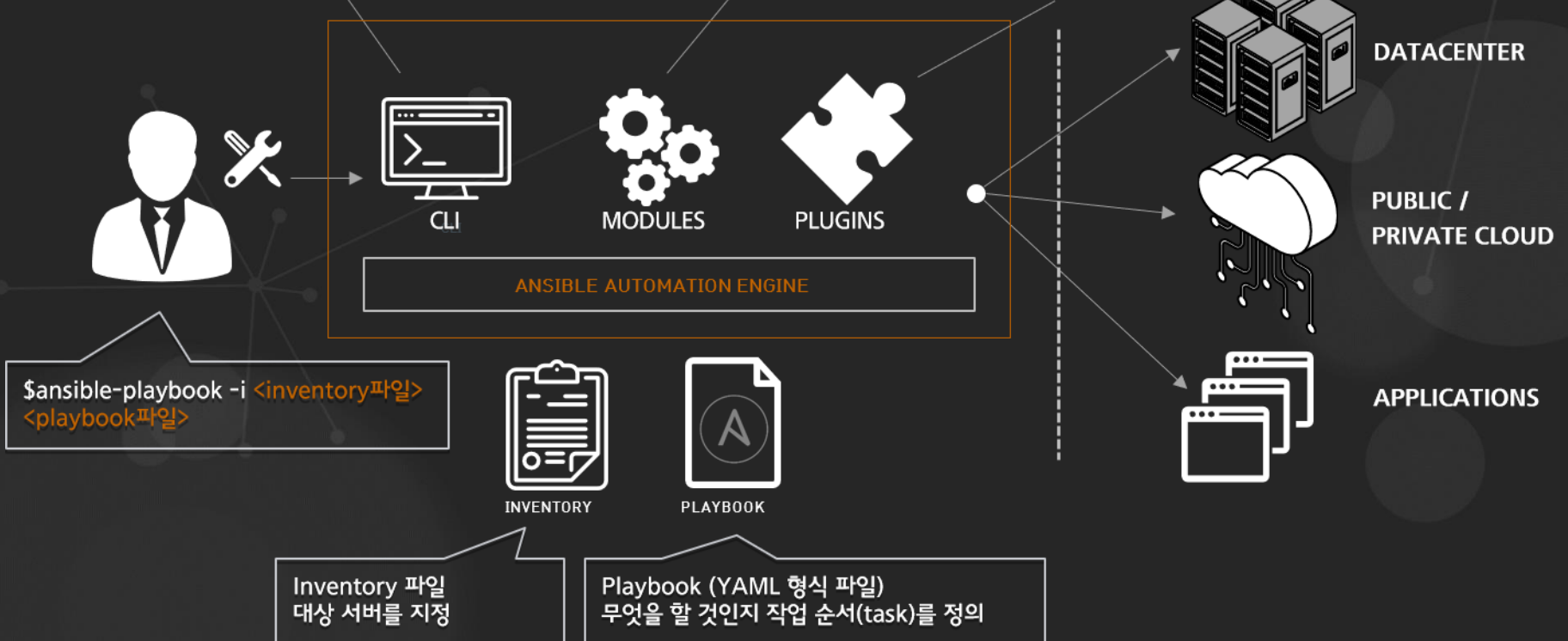
# Ansible 핵심 컴포넌트

- 대상 호스트에 Python 필요 ( Python 2.4 이상 )
- 사용자는 inventory 파일과 Playbook 파일 만 작성

Ansible를 조작하기위한 인터페이스

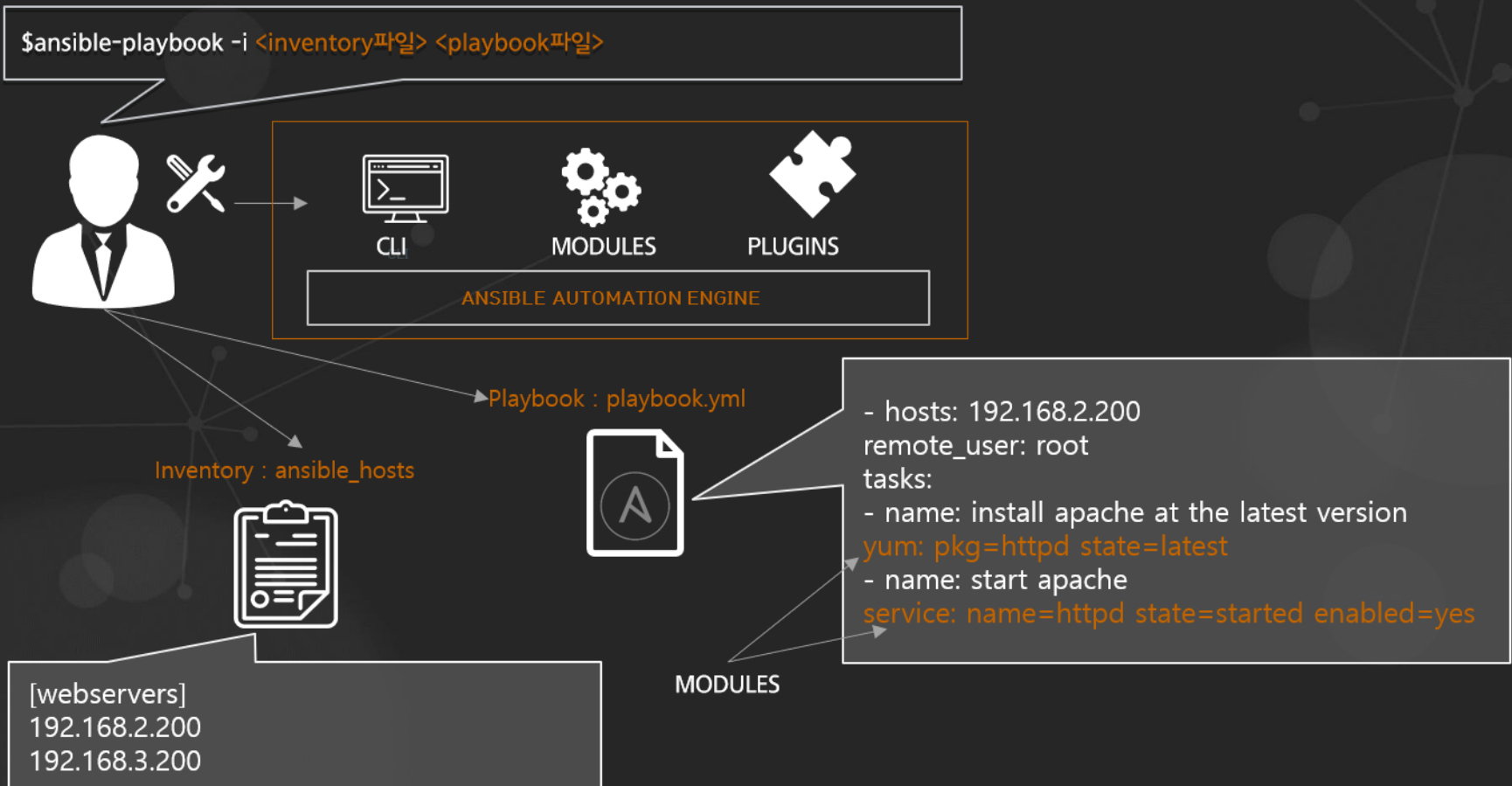
재사용 가능한 처리 장치. 작업을 포장 한 구성 요소

Ansible의 핵심 기능을 확장 하기위한 구성 요소



# Ansible 사용 방법

- 사용자는 inventory 파일과 Playbook 파일 만 작성
- “192.168.2.200” 서버에 대해서 root 사용자로 yum 을 사용해 Apache Httpd 를 설치 하고 Apache Httpd 을 실행



# ansible-playbook 커멘드 실행

1 `$ ansible-playbook -i ansible_hosts playbook.yml`



SSH



3

```
# yum install httpd
# service httpd start
# chkconfig httpd on
(OR systemctl enabled httpd)=0
```

구성되는 서버  
대상 서버에는 에이전트는 필요 없으나  
Python이 설치 되어야함.

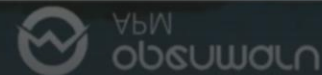
```
PLAY [192.168. 2.200]
*****

GATHERING FACTS
*****
ok: [192.168. 2.20]
:
:

PLAY RECAP
*****
192.168. 2.20 : ok=2  changed=2  unreachable=0  failed=0
```

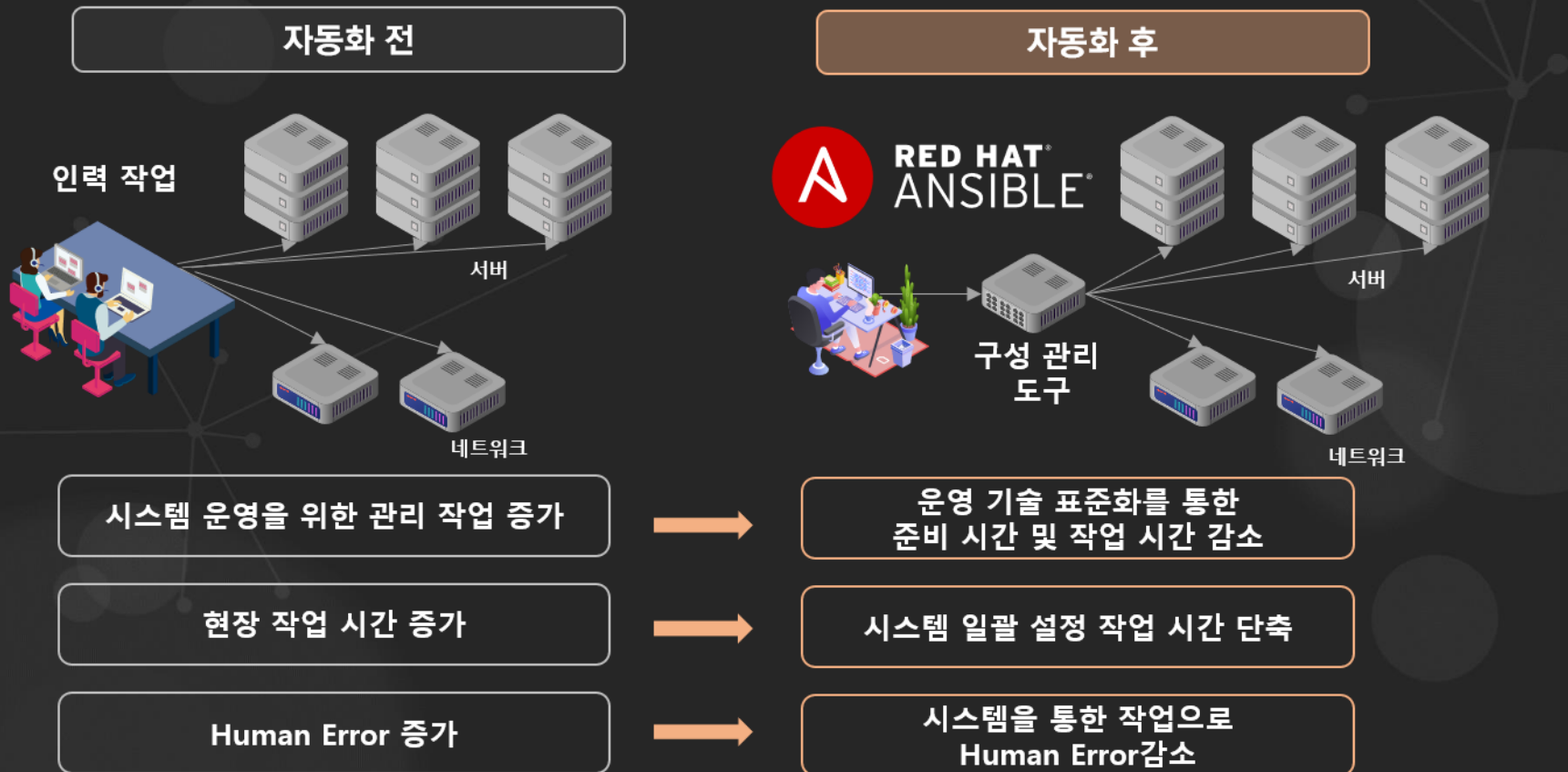
# Application Performance Management

## Ansible 적용 효과



# Ansible 을 통한 IT 인프라 운영 자동화

- IT 인프라의 대규모화, 가상화 및 고도화에 따라 IT 장비에 대한 환경설정 및 정보 취합이 복잡하고 어려움
- 작업 계획시간과 현장 작업 시간의 증가와 휴먼 에러의 증가



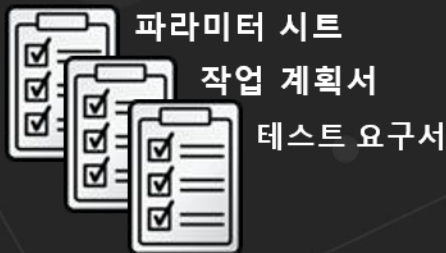


# Ansible 자동화 추진 방안

- Start Small Think Big

단순한 작업들을 대체  
(개발이 쉬운 부분부터 일부분 자동화)

대체



Playbook



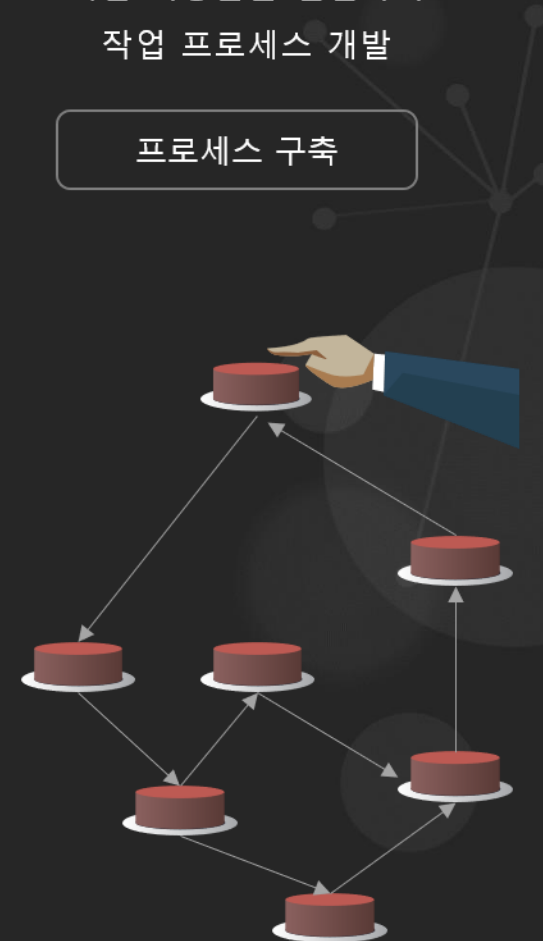
자동화를 통하여  
운영 작업 위탁 ( 버튼 )

기능화 (서비스화)



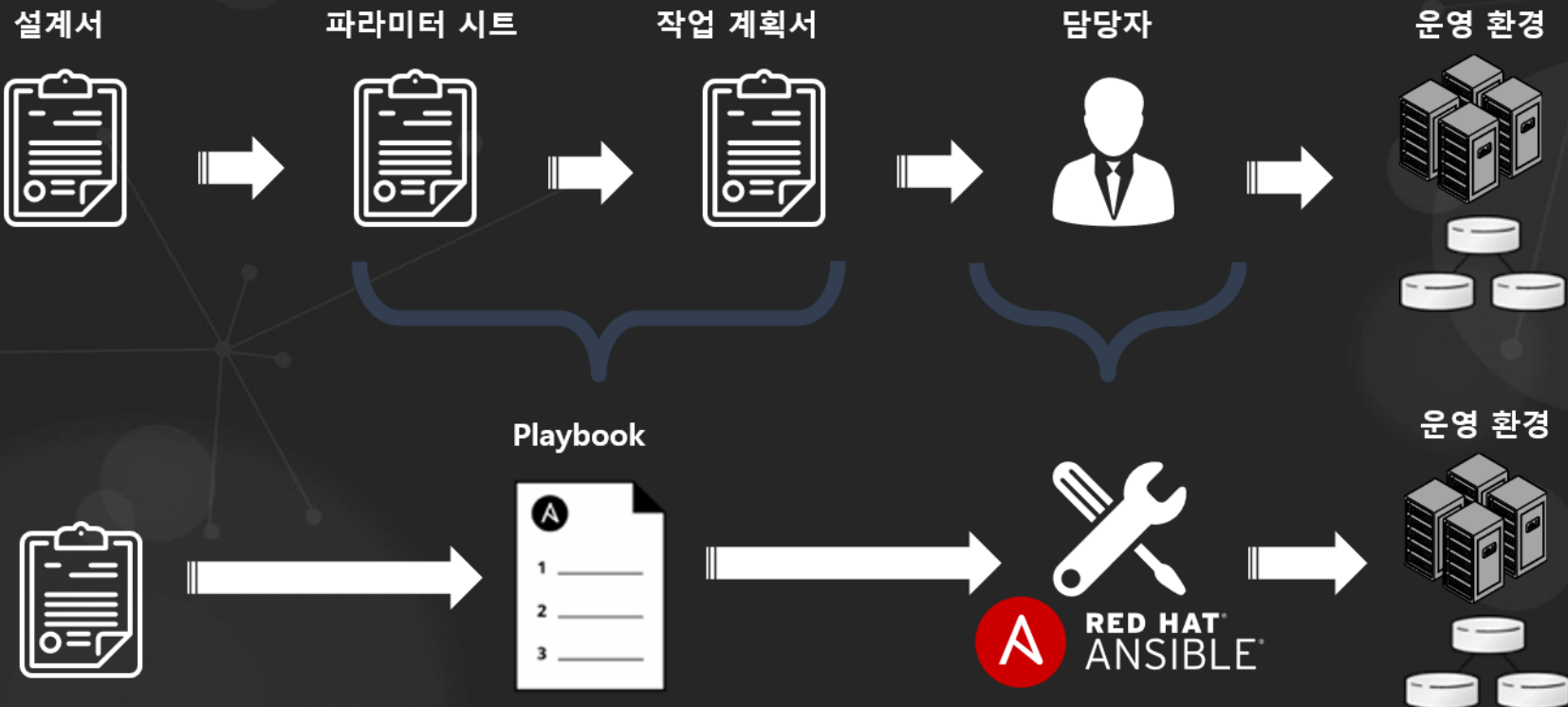
작은 기능들을 연결하여  
작업 프로세스 개발

프로세스 구축



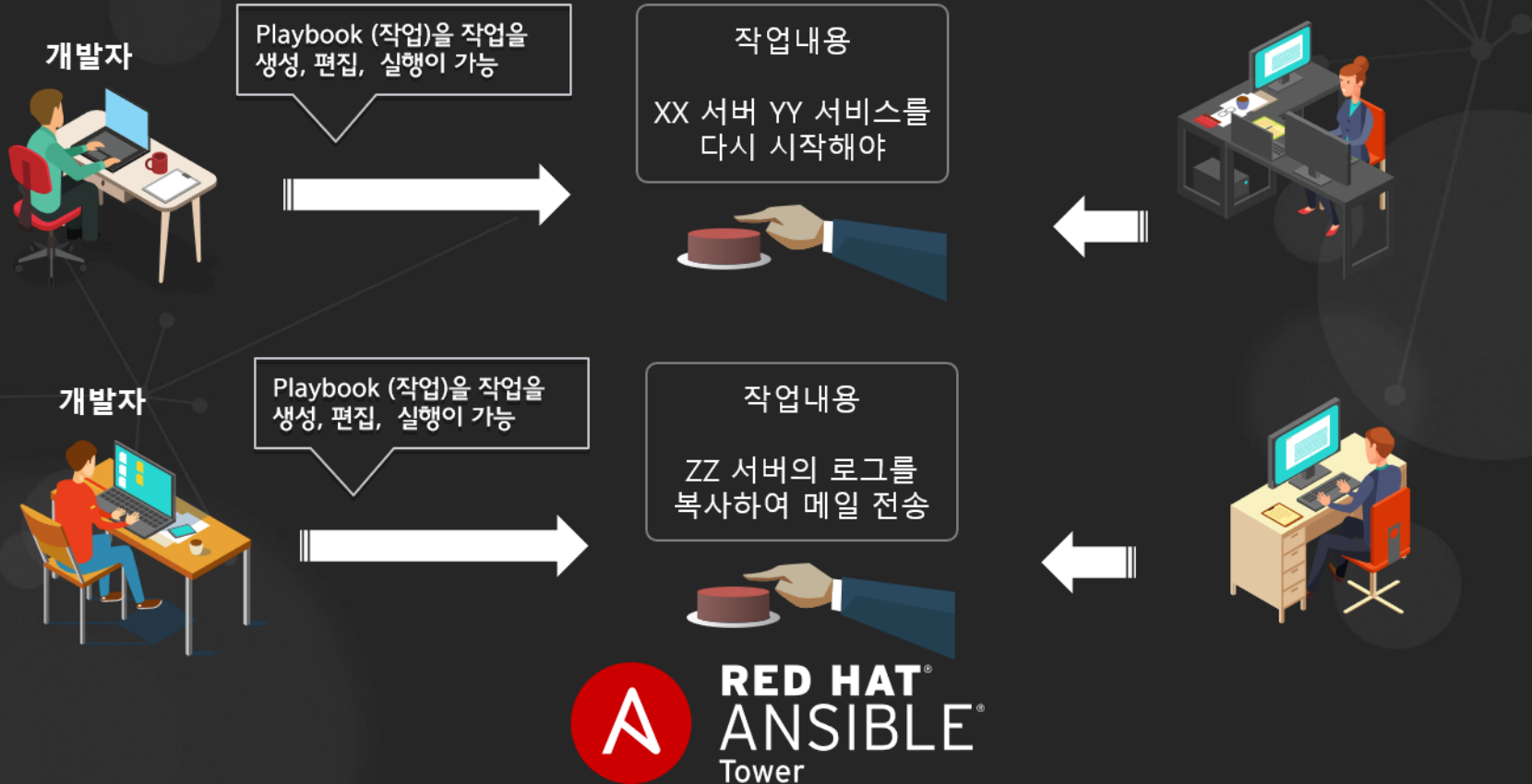
# Ansible 을 통한 작업 계획서 대체

- 현재 운영 작업들을 Playbook 로 정의하여 자동화
  - Playbook 은 실행 가능한 파라미터 시트 겸 가이드



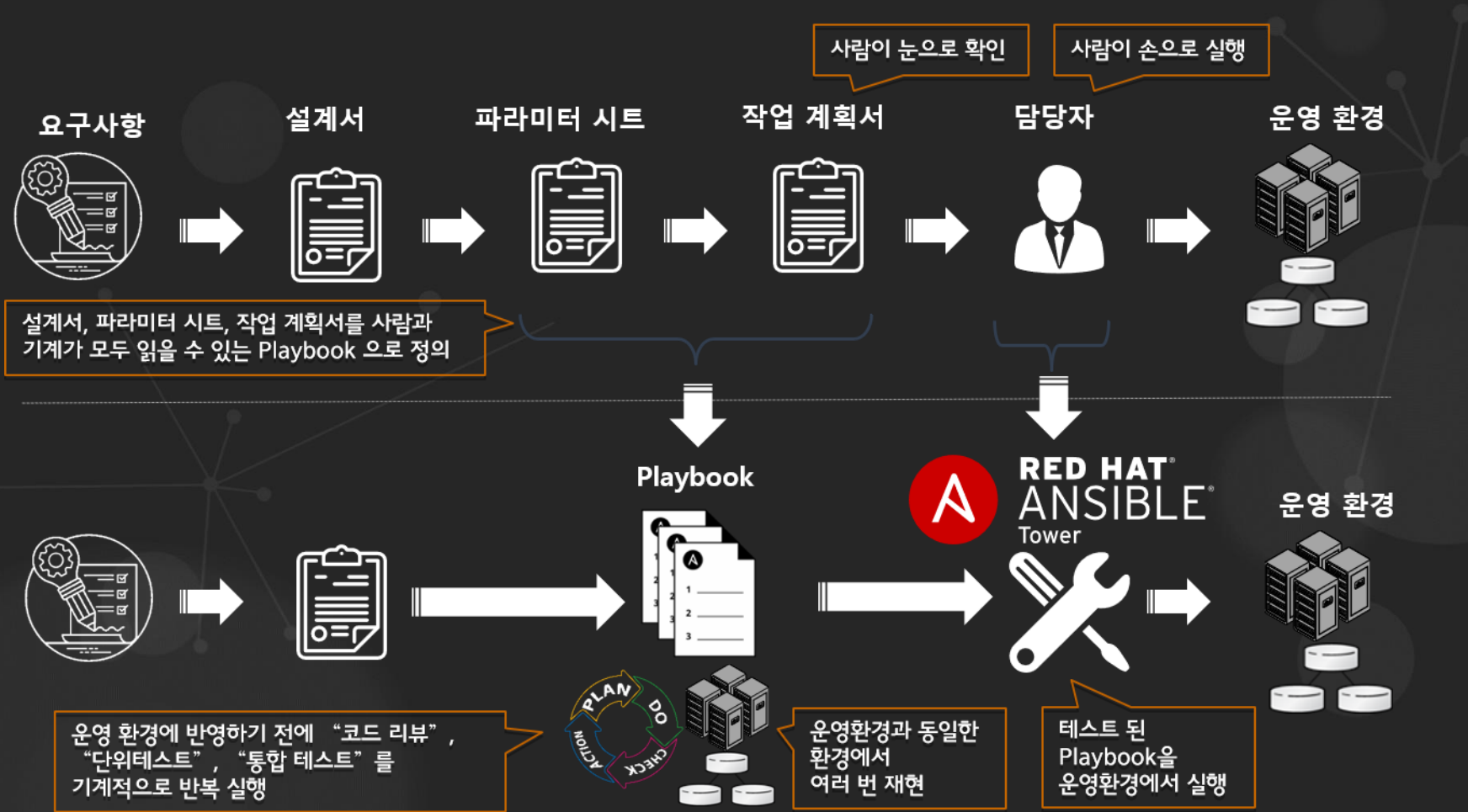
# 운영 작업 위임 (서비스 화)

- 개발된 Playbook 에 대한 실행 권한을 운영자에게 위임
  - 서비스를 재 시작할 수 있는 권한을 운영자에게 위임
  - 서버 로그를 메일로 전달하는 권한을 운영자에게 위임



# Ansible 을 통한 전체 작업의 효율화

- Ansible Playbook으로 Code를 정의하고 검증 후 운영 환경에서 반영




# 애플리케이션을 위한 인프라 제공


- 클라우드 환경에서 앤서블을 통한 인프라 제공의 효과


작업	시간 (분)
가상머신 생성	30
스토리지/네트워킹 추가	30
팀 (작업) 전환 시 대기 시간	120
운영체제 설치	60
설치 후 대기	60
운영체제 구성	90
응용프로그램 설치	90
응용프로그램 구성	45
팀 (작업) 전환 시 대기 시간	120
보안 구성 및 검사	90
실제 작업 시간	5 시간 15분
전체 시간	10 시간

  
**앤서블 오케스트레이션**

작업	시간 (분)
가상머신 생성	2
스토리지/네트워킹 추가	3
<del>팀 (작업) 전환 시 대기 시간</del>	<del>120</del>
운영체제 설치	2
<del>설치 후 대기</del>	<del>60</del>
운영체제 구성	1
응용프로그램 설치	2
응용프로그램 구성	1
<del>팀 (작업) 전환 시 대기 시간</del>	<del>120</del>
보안 구성 및 검사	2
실제 작업 시간	0 분
전체 시간	13 분


**운영 자동화  
(인프라 프로비저닝)**


**형상 관리  
+ 운영 자동화**

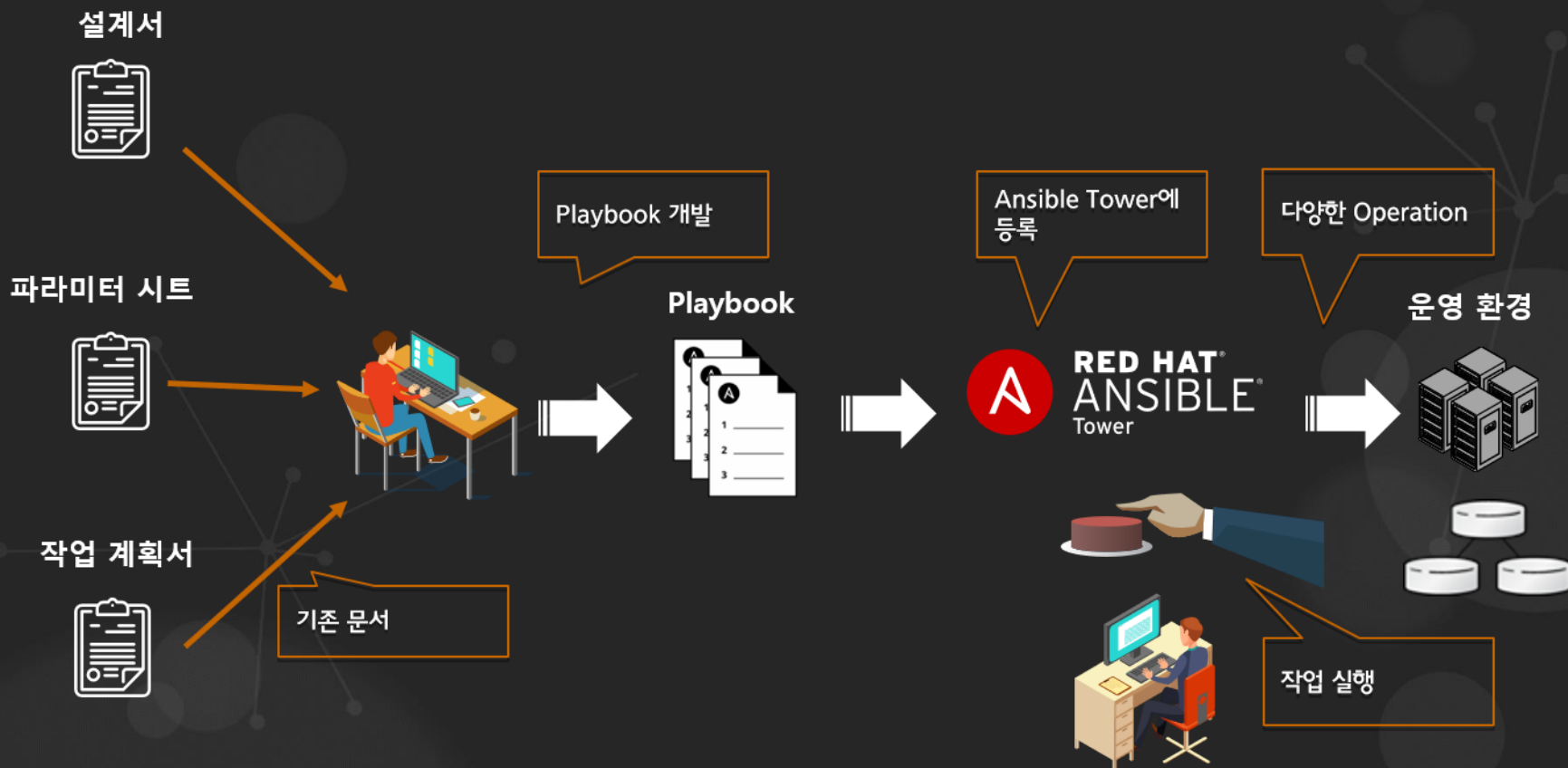

**구성 진단**

**Application Performance Management**



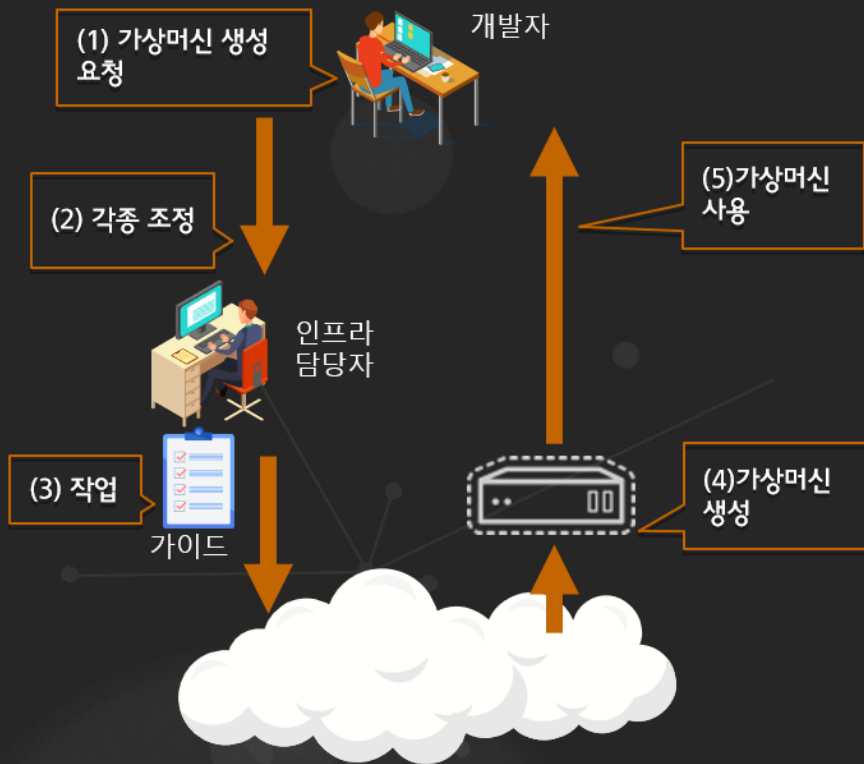
**Ansible Use Case**

# Ansible 에서의 자동화의 흐름



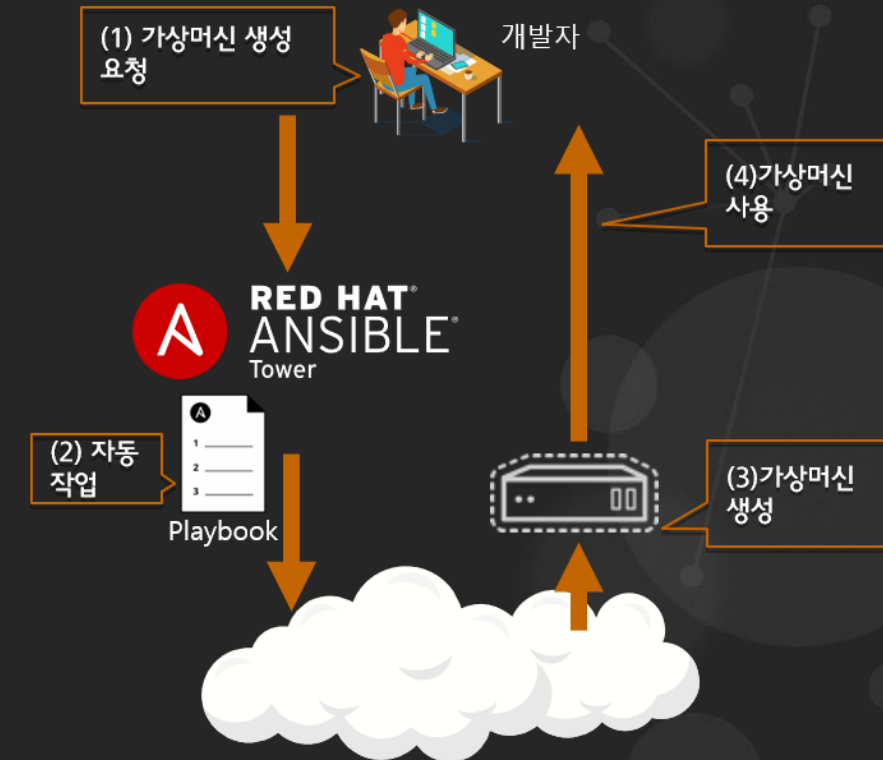
# Ansible 도입 방안 – 가상화

## Before



- ✓ 사람에 의한 조정과 작업
- ✓ Human Error 와 품질의 차이

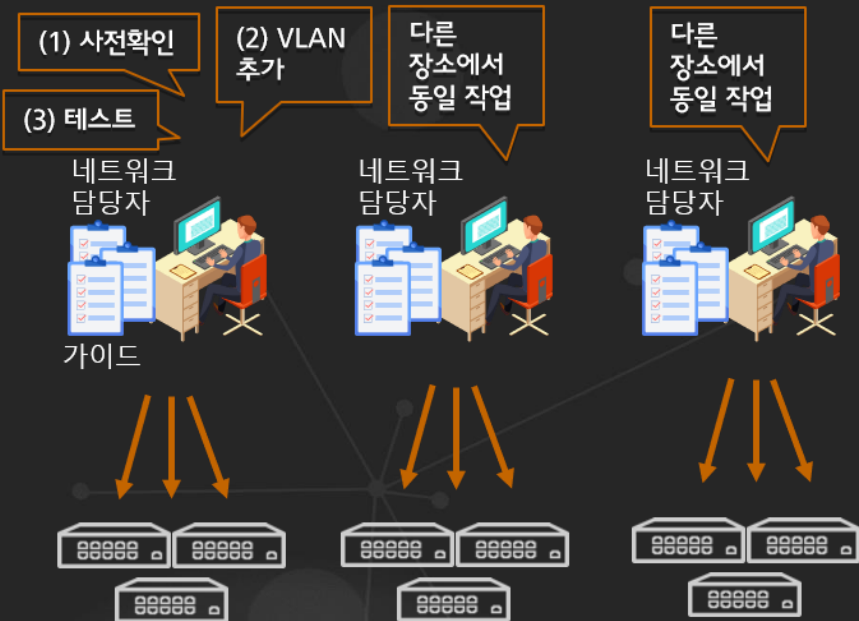
## After



- ✓ 테스트까지도 자동화
- ✓ 일정한 품질 유지

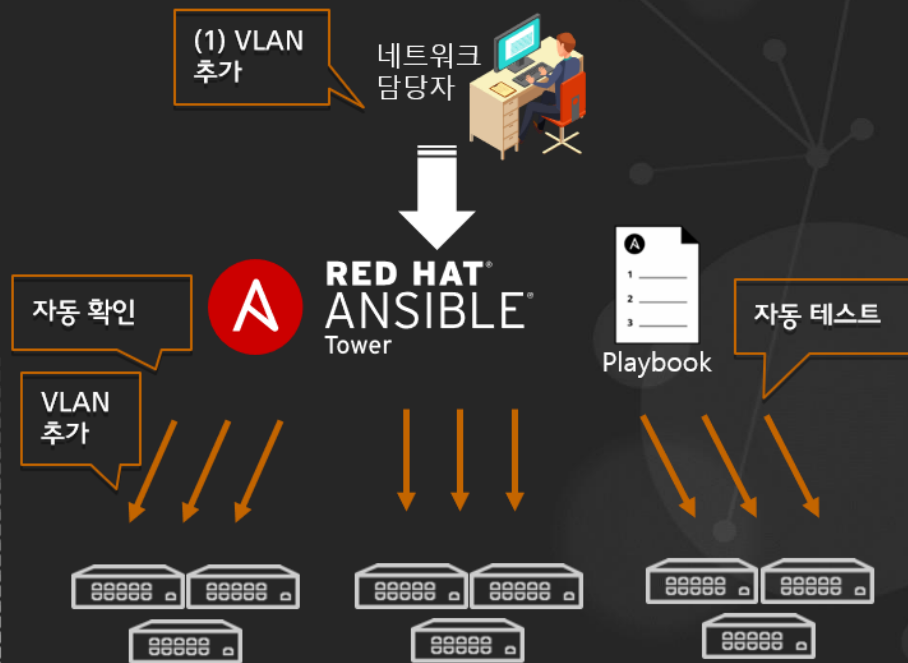


## Before



- ✓ 동일한 작업을 반복 수행, 횟수 만큼 시간 소요
- ✓ 제한적인 테스트와 품질의 차이

## After



- ✓ 하나를 자동화하면 반복적인 작업 가능
- ✓ 자동으로 확인하고 테스트
- ✓ 일정한 품질 유지

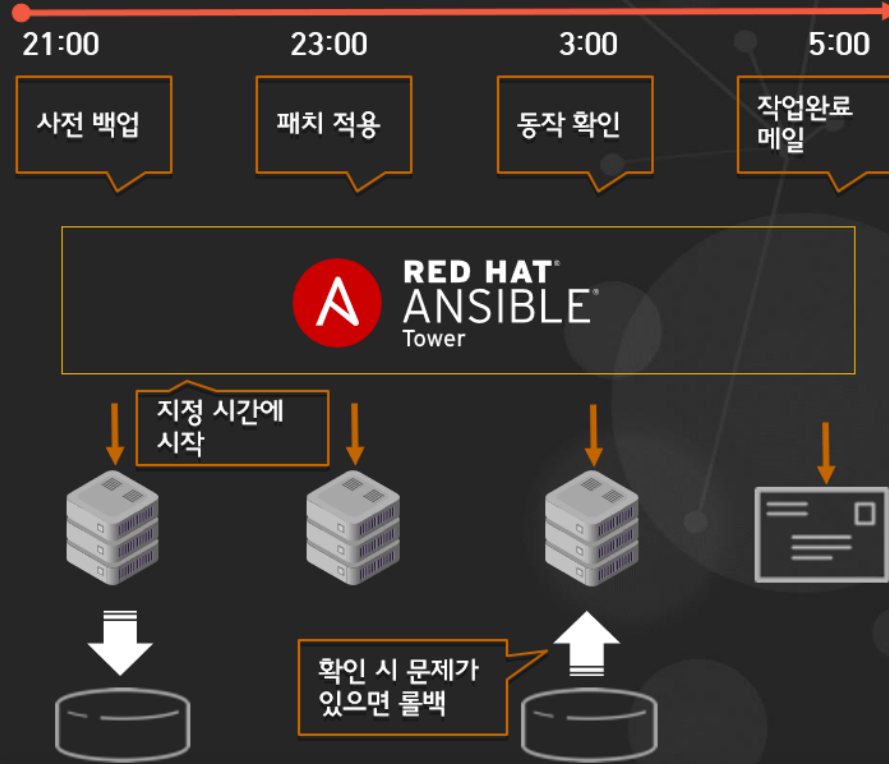
# Ansible 도입 방안 – 야간 패치 작업

## Before



- ✓ 작업하는 동안 상주 지원하며 시간 소요
- ✓ 오류 확인 및 롤백에 대한 위험요소
- ✓ 품질의 차이

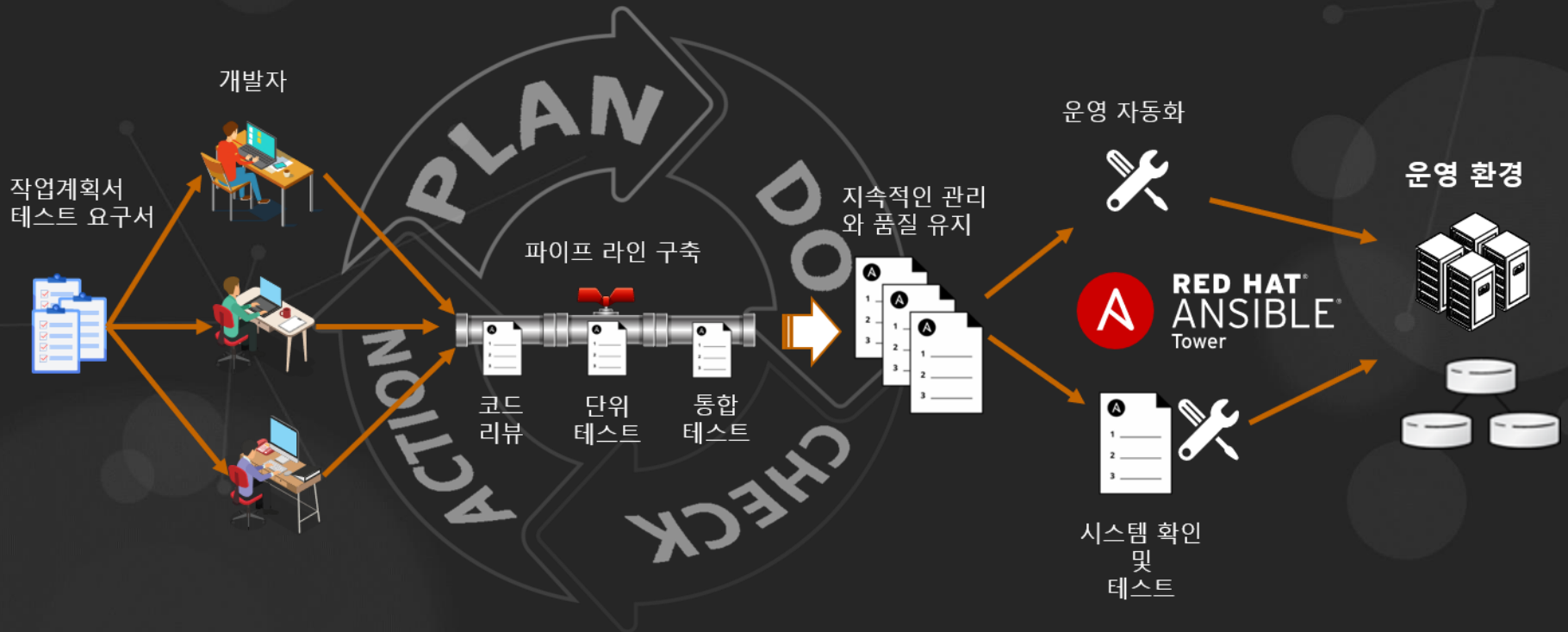
## After



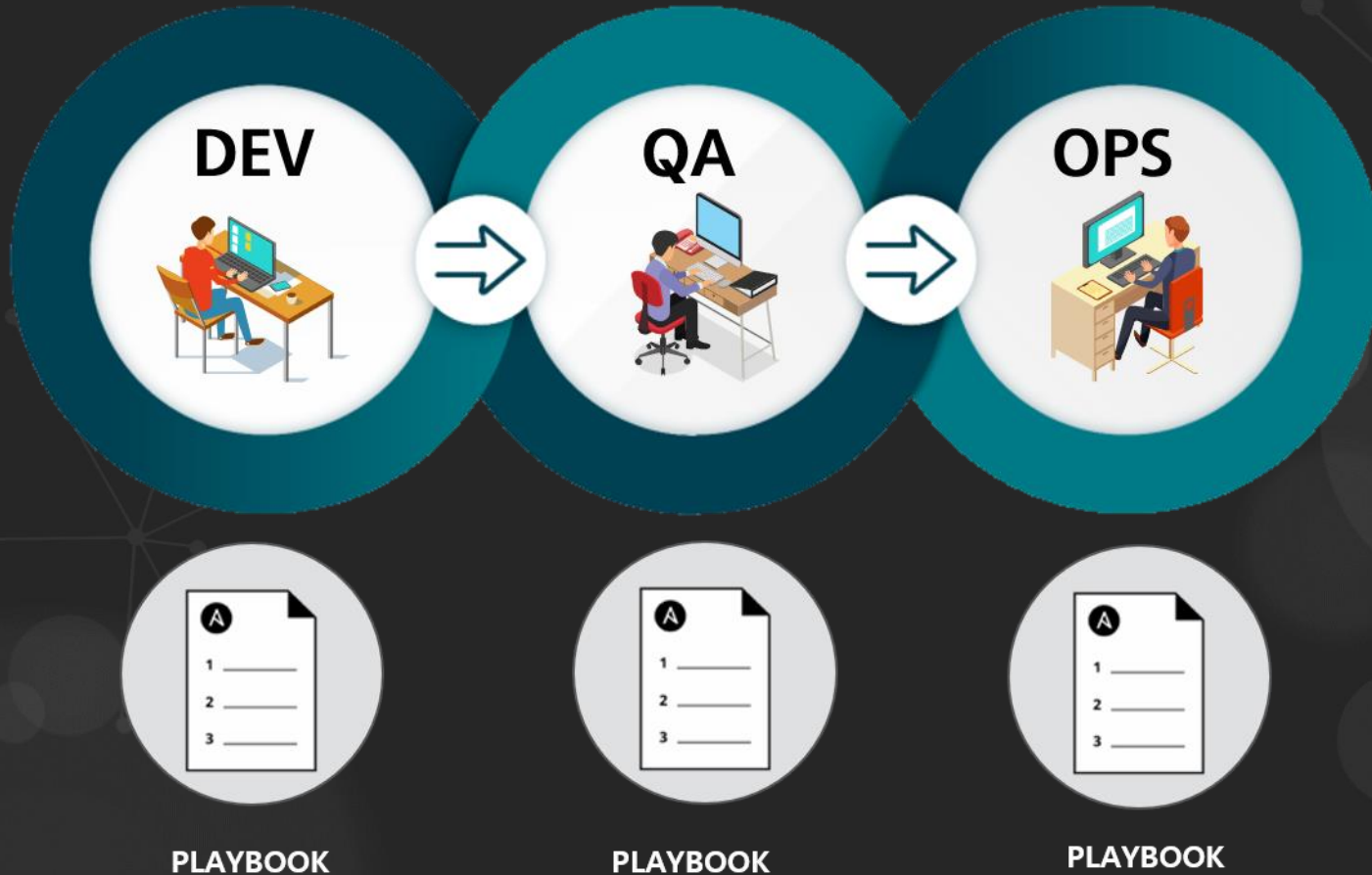
- ✓ 비상주 지원과 유비보수 시간 단축
- ✓ Human Error 차단
- ✓ 일정한 품질 유지

# Ansible 도입 방안 - 인프라 CI 구현

- 산출물( Playbook )의 품질을 관리하고 지속적인 관리
  - Infrastructure as Code 구현
  - 각종 CI / CD 도구와 함께 적용



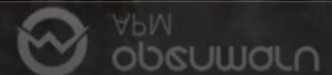
# Dev+Ops를 실현하는 Ansible



# Application Performance Management

**“ IF YOU CAN'T MEASURE IT  
YOU CAN'T MANAGE IT. ”**

*- Peter Drucker*



Application Performance Management

감사합니다.



openmaru  
APM



APM  
openmaru



제품 / 서비스에 관한 문의

- 콜 센터 : 02-469-5426 ( 휴대폰 : 010-2243-3394 )
- 전자 메일 : [sales@openmaru.com](mailto:sales@openmaru.com)