01. Introduction and Shell Scripting

Introduction

What is DevOps?

DevOps is a methodology in the software development and IT industry. Used as a set of practices and tools, DevOps integrates and automates the work of software development (Dev) and IT operations (Ops) as a means for improving and shortening the systems development lifecycle.

There are two types of **people involved**:

- **Developers:** highly interested in improving the system. Often willing to experiment and make disruptive changes. There's a cultural acknowledgement that this is the case (move fast and break things), but this is not always a good idea.
- **Operations:** prioritise maximizing uptime above all else. Highly sceptical of major changes, and view them as a disruptive. Historically, this has put them into conflict with developers.
- **Other Roles:** there are many other types of people in a typical software development organisation (security experts, management, software testers, ...).

DevOps

DevOps aims to overcome tensions between the people by utilising technologies that allow these roles to work together. This technologies are very useful for making software systems work better in general.

How do we do this?

- An agile software development process applied to the entire software lifecycle.
- Extensive use of collaborative tools.

- Transparency between groups and teams.
- Extensive use of virtualisation, containers, infrastructure as code and related technologies.

Technologies

- **Shell Scripting - BASH**
- **Version Control - git**
- **Virtualisation - VirtualBox**
- **Cloud Computing - AWS**
- **Containerisation - docker**
- **Infrastructure as Code - Terraform**
- **Continuous Integration - Jenkins**

CA(L)MS

Culture, automation, measurement, lean (optional), sharing.

- Every letter implicitly includes the lean, that’s why its optional.

Culture

DevOps implies a fundamental change in the way that a company or organisation works, with extensive use of agile and collaboration.

Automation

The setup and maintenance of both development and operations/deployment platforms should be as automated as possible.

Lean

The adoption of Lean thinking, putting an emphasis on efficiency and implementing organisation-wide changes to facilitate this.

Measurement

Everything that can be measured should be. Measurements are used to guide decisions: do what is shown to work, not what you think will work.

Sharing

Everyone involved should have access to all the code and infrastructure.

Bash Scripting

Operating System Basics

- **Shell:** user interface
- **Kernel:** manages the software
- In modern systems there is a middle layer for the c compiler, init program/sequence, ...

Bash

```
echo Hello, World! # expands variables

echo "Hello, World!" # expands variables

echo 'Hello, World!' # does not expand variables

chmod +x script.sh # to give permissions on the shell

# No need to use #! /bin/bash but it is considered good pra
ctice. Bash executes in bash by default
```

```
#! /bin/bash
# Printf example

PI=3,14159265359 # if there are spaces (PI = 3,1415) it con
siders PI is a program and the following are its arguments,
so no spaces.
printf "The value of pi is %.2f\n" $PI
```

```
#! /bin/bash
# String comparisons

if [ "foo" == "bar" ]; then
    echo "foo equals bar"
    echo "this should never happen"
else
    echo "foo does not equal bar"
fi

# to compare numbers
if [ 1 -eq 1 ]; # use eq (equals)
```

```
#! /bin/bash
# Number comparisons

if [ $1 -le $2 ]; then
    echo "$1 is less than or equal to $2"
else
    echo "No"
fi
```

```
#! /bin/bash
# One digit number check (multiple conditions)

if [ $1 -ge 0 ] && [ $1 -lt 10 ]; then
    echo "I was given a single digit number"
fi
```

```
#! /bin/bash
# While loop example

NUMBER=$1
if [ -z $NUMBER ]; then # - checks for empty string
    >&2 echo "You forgot a number" # >&2 redirects output t
o stderr
```

```
    exit 1
fi

let VAR=0
while [ $VAR -le $NUMBER ]; do
    echo $VAR
    let VAR++ # let lets you do math operations (i.e. incre
ments)
done
```

```
#! /bin/bash
# For loop example

NUMBER=$1
if [ -z $NUMBER ]; then # - checks for empty string
    >&2 echo "You forgot a number" # >&2 redirects output t
o stderr
    exit 1
fi

for (( i=0; i<$NUMBER; i++)); do
    echo $i
done
```

```
#! /bin/bash
# Functions in Bash

# defining the function
say_hello() {
    NAME=$1
    echo "Hello, $NAME"
}

# calling the function
say_hello "Elena"
```

```
#! /bin/bash
# Functions that return values

multiply() {
    local ONE=$1 # variable only in the scope of the functi
on
    local TWO=$2

    RESULT=$(( ONE * TWO )) # varible in the scope of the p
rogram
}

multiply 5 2
echo "5 times 2 is $RESULT"
# if result was defined with local it wouldn't be printed w
ith the echo!
```

```
#! /bin/bash
# Functions that return values

multiply() {
    local ONE=$1 # variable only in the scope of the functi
on
    local TWO=$2

    local RESULT=$(( ONE * TWO )) # varible in the scope of
the program
    echo $RESULT
}

N=$(multiply 5 2) # the result is stored as data in bash wh
en using $() ex: $(ls)  stores the result of ls as data.
echo "5 times 2 is $N"
```

```
#! /bin/bash
# For each loop
```

```
for i in {0..10}; do
    echo $i
done
```

```
#! /bin/bash
# Arrays in Bash

SHOPPING_LIST=("bananas" "apples" "milk")

for item in ${SHOPPING_LIST[@]}; do
    echo "We need to buy $item"
done
```

```
#! /bin/bash
# Associative Arrays (key-value pairs) in Bash

declare -A ages
ages=(
    [John]=20 # key John maps to value 20
    [Mary]=23
    [Bob]=60
    [Anne]=43
)

# order can change every time you run it (order is not guar
anteed)
for name in ${!ages[@]}; do
    echo "$name is ${ages[$name]} years old"
done
```