**Immutable infrastructure**

An immutable infrastructure is another infrastructure paradigm in which servers are never modified after they’re deployed. If something needs to be updated, fixed, or modified in any way, new servers built from a common image with the appropriate changes are provisioned to replace the old ones. After they’re validated, they’re put into use and the old ones are decommissioned.

**Example**- let say you have setup infrastructure with Java 7 and tomcat, and now you want to modify it to java8 and tomcat 8, so you will be setting up a new environment, and once setup is completely up and running then you will be decom previous infrastructure.

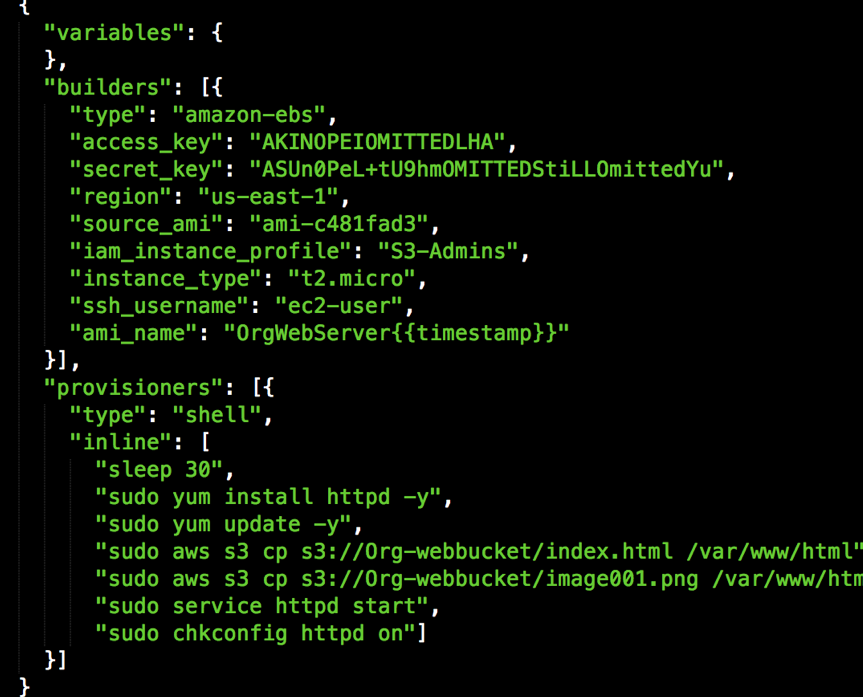
Difference between IT infra and devops? In IT infra we will go for troubleshooting whereas in devops infra we will create immutable infra.

In which language packer is developed? Go (incorrectly known as Golang, is a statically typed, compiled programming language designed at Google by Robert Griesemer, Rob Pike, and Ken Thompson. Go is syntactically similar to C, but with memory safety, garbage collection, structural typing.

Any other tool that developed in GO- terraform, docker and Kubernetes.

Advantage of GO- They give similar output like C, and high-level language like Java.

Packer template-



Builder- it will tell you where we need to generate the image.

Provisioners- Builder create your image, and provisioner logged-in to that system, and install all the required software.

Post processor- They will give the final touch like integration with other services, primary function is compression(if file size is more than zipping it) and conversions(from Linux to window image).

**JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language.

Object- Anything that represent real time data, called object.

JSON JSON: **J**ava**S**cript **O**bject **N**otation. JSON is a syntax for storing and exchanging data. JSON is text, written with JavaScript object notation.

**Exchanging Data** When exchanging data between a browser and a server, the data can only be text. JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server. We can also convert any JSON received from the server into JavaScript objects. This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

**Sending Data** If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

**Example** var myObj = {name: "John", age: 31, city: "New York"};  
var myJSON = JSON.stringify(myObj);  
window.location = "demo\_json.php?x=" + myJSON;

**Receiving Data** If you receive data in JSON format, you can convert it into a JavaScript object:

**Example** var myJSON = '{"name":"John", "age":31, "city":"New York"}';  
var myObj = JSON.parse(myJSON);  
document.getElementById("demo").innerHTML = myObj.name;

**Storing Data**

When storing data, the data has to be a certain format, and regardless of where you choose to store it, *text* is always one of the legal formats.JSON makes it possible to store JavaScript objects as text.

Storing data in local storage

// Storing data:  
myObj = {name: "John", age: 31, city: "New York"};  
myJSON = JSON.stringify(myObj);  
localStorage.setItem("testJSON", myJSON);  
  
// Retrieving data:  
text = localStorage.getItem("testJSON");  
obj = JSON.parse(text);  
document.getElementById("demo").innerHTML = obj.name;

**What is JSON?**

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is a lightweight data-interchange format
* JSON is "self-describing" and easy to understand
* JSON is language independent **\***

\*  
JSON uses JavaScript syntax, but the JSON format is text only.  
Text can be read and used as a data format by any programming language.

The JSON format was originally specified by [Douglas Crockford](http://www.crockford.com).

**Why use JSON?**

Since the JSON format is text only, it can easily be sent to and from a server and used as a data format by any programming language.

JavaScript has a built-in function to convert a string, written in JSON format, into native JavaScript objects:

JSON.parse()

So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object

The JSON syntax is a subset of the JavaScript syntax.

## JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

* Data is in name/value pairs
* Data is separated by commas
* Curly braces hold objects
* Square brackets hold arrays

## JSON Data - A Name and a Value

JSON data is written as name/value pairs.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

### Example

"name":"John"

JSON names require double quotes. JavaScript names don't.

## JSON - Evaluates to JavaScript Objects

The JSON format is almost identical to JavaScript objects. In JSON, keys must be strings, written with double quotes:

### JSON { "name":"John" }

## JSON Values

In **JSON**, values must be one of the following data types:

* a string
* a number
* an object (JSON object)
* an array
* a boolean
* null

In **JavaScript** values can be all of the above, plus any other valid JavaScript expression, including:

* a function
* a date
* undefined

In JSON, string values must be written with double quotes:

### JSON

{ "name":"John" }

In JavaScript, you can write string values with double or single quotes:

# JSON vs XML

Both JSON and XML can be used to receive data from a web server.

The following JSON and XML examples both define an employees object, with an array of 3 employees:

### JSON Example

{"employees":[  
  { "firstName":"John", "lastName":"Doe" },  
  { "firstName":"Anna", "lastName":"Smith" },  
  { "firstName":"Peter", "lastName":"Jones" }  
]}

### XML Example

<employees>  
  <employee>  
    <firstName>John</firstName> <lastName>Doe</lastName>  
  </employee>  
  <employee>  
    <firstName>Anna</firstName> <lastName>Smith</lastName>  
  </employee>  
  <employee>  
    <firstName>Peter</firstName> <lastName>Jones</lastName>  
  </employee>  
</employees>

## JSON is Like XML Because

* Both JSON and XML are "self describing" (human readable)
* Both JSON and XML are hierarchical (values within values)
* Both JSON and XML can be parsed and used by lots of programming languages
* Both JSON and XML can be fetched with an XMLHttpRequest

## JSON is Unlike XML Because

* JSON doesn't use end tag
* JSON is shorter
* JSON is quicker to read and write
* JSON can use arrays

The biggest difference is:

 XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

## Why JSON is Better Than XML

XML is much more difficult to parse than JSON.  
JSON is parsed into a ready-to-use JavaScript object.

For AJAX applications, JSON is faster and easier than XML:

Using XML

* Fetch an XML document
* Use the XML DOM to loop through the document
* Extract values and store in variables

Using JSON

* Fetch a JSON string
* JSON.Parse the JSON string

## Valid Data Types

In JSON, values must be one of the following data types:

* a string
* a number
* an object (JSON object)
* an array
* a boolean
* null