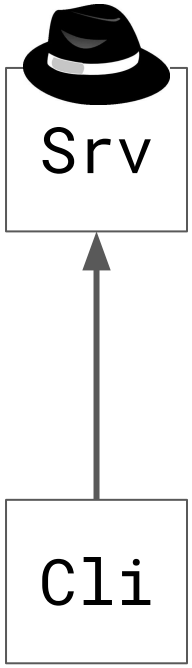
ETHEREUM DEV BARCELONA Meetup

µtraining 4 devs v2

adria@codecontext.io

PART I

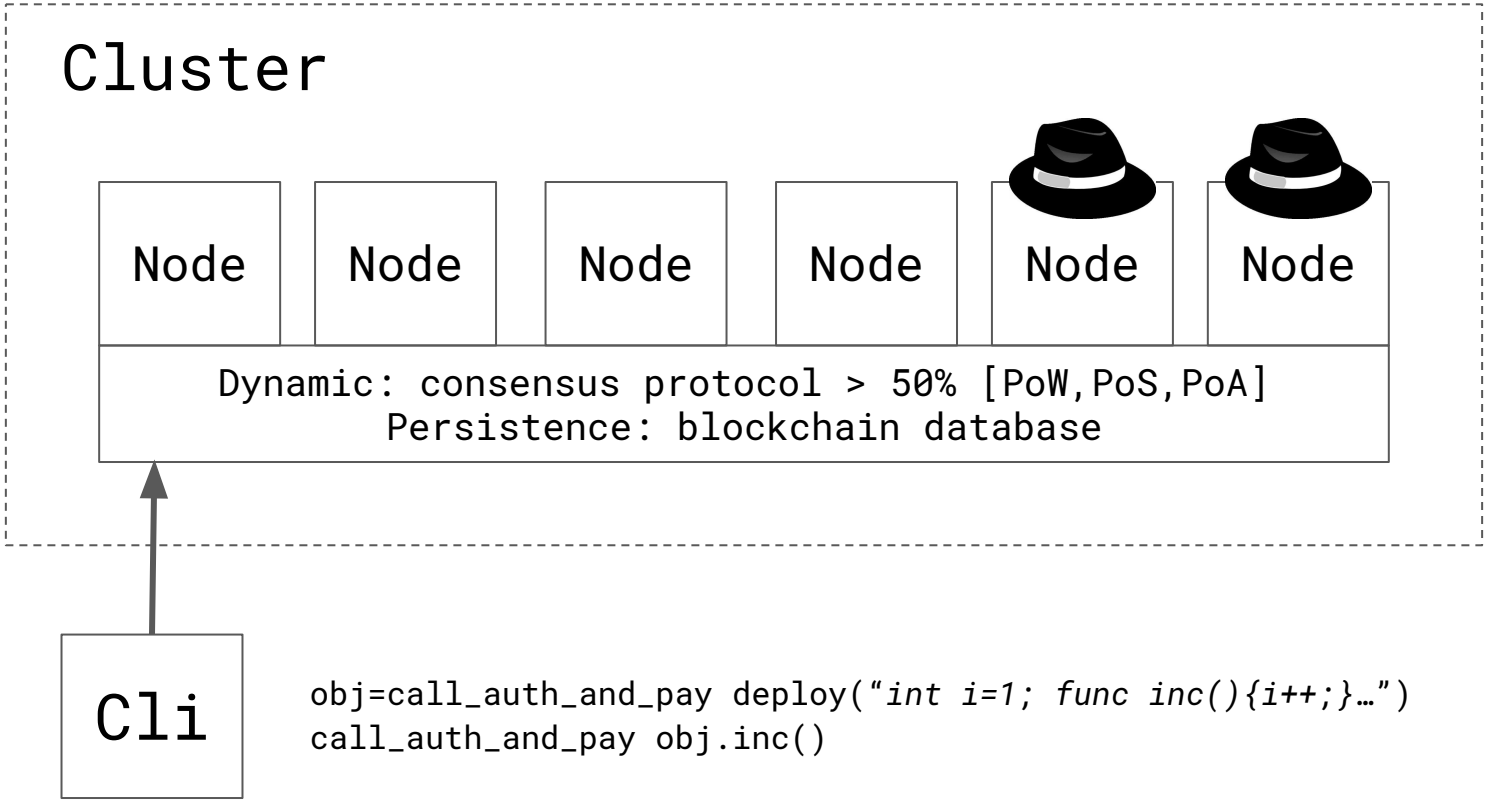
ethereum




```
deploy code in server

int i=1
func inc() {
  i++;
}
func get() {
  return i;
}
```

```
call obj.inc()
```



https://ethstats.net/

- It's a **secure**, **append-only database** maintained by **peers consensus protocol**
- Append operations are done each 15 seconds, so writes are grouped with blocks
- Stores:
  - Smartcontract code
  - Calls done to the smartcontracts
  - Ether transfer between addresses
- State is NOT stored (it's calculated and stored in nodes)

https://ropsten.etherscan.io/

```
pragma solidity ^0.4.8;
contract Counter {
   uint i=1;
   function inc() {
      i++;
   }
   function get() constant returns (uint) {
      return i;
   }
}
```

- You pay in an execution of a method of an object
  - Database usage
  - Memory usage
  - CPU consumption (**GAS** unit)
- You pay in an embedded cryptocurrency: **ETHER**
  - ETHERS are a reward for servers maintaining the network
  - You buy server's ETHERS in a market like coinbase
  - You specify ETHERS per GAS that you want to pay
  - I will pay 0.0002 ETHERS/GAS in this function
- Ether accounts are called **ADDRESS** (like Bank IBAN)
  - And transfer them from an account to another
  - In some implementations (nets) has a market value
  - Two type
    - Wallets : personal accounts with a private key
    - **SMARTCONTRACTS**

https://ethereum.github.io/browser-solidity/

PART II

solidity

bool	true, false
uint	alias for uint256
int	alias for int256
address	
string	"ETHS GO!"
enum	enum State { Open, Closed } ; State u = State.Open;
struct	struct Point { uint x, uint y}; Point p1,p2;
mapping	mapping (string => int) ages; ages["juan"]=12; ages["pepe"]=60
vector	int[] v; v.push(1); v[0]

All composite types (struct, mapping, vector) are references, so data will be not copied
Not needed to initialize, default values are applied

Base code

```
pragma solidity ^0.4.8;
contract RRLottery {
  uint i;

  function get() returns (uint) { return i; }
  function inc() {
     if (i==5) {
        i=0;
    } else { i++; }
  }
}
```

constructor

```
pragma solidity ^0.4.8;
contract RRLottery {
  uint i;
  uint bet;

  function RRLottery(uint _bet)     { bet = _bet; }
  function get() returns (uint)     { return i; }
  function inc() {
     if (i==5) {
        i=0;
    } else { i++; }
  }
}
```

address && msg.sender

```
pragma solidity ^0.4.8;
contract RRLottery {
  uint i;
  uint bet;
  address owner;

  function RRLottery(uint _bet)          { bet = _bet; owner = msg.sender; }
  function get() returns (uint)          { return i; }
  function inc() {
     if (i==5) {
        i=0;
    } else { i++; }
  }
  function reset() {
     if (msg.sender != owner) throw;
     i = 0;
  }
}
```

```
pragma solidity ^0.4.8;
contract RRLottery {
  uint i;
  uint bet;
  address owner;

  function RRLottery(uint _bet)          { bet = _bet; owner = msg.sender; }
  function get() constant returns (uint) { return i; }
  function inc() {
     if (i==5) {
        i=0;
    } else { i++; }
  }
  function reset() {
     if (msg.sender != owner) throw;
     i = 0;
  }
}
```

payable, address.send, msg.value

```
pragma solidity ^0.4.8;
contract RRLottery {
  uint i;
  uint bet;
  address owner;

  function RRLottery(uint _bet)          { bet = _bet; owner = msg.sender; }
  function get() constant returns (uint) { return i; }
  function inc() payable {
     if (msg.value < bet) throw;
     if (i==5) {
        msg.sender.send(this.balance);
        i=0;
    } else { i++; }
  }
  function reset() {
     if (msg.sender != owner) throw;
     owner.send(this.balance);
     i = 0;
  }
}
```

Small smartcontract

- Make an exercise together
- I lend you a tool (e.g. Hammer)
  - but I want you to make a security deposit of at least 1000 wei
- When returned
  - if both agree, the deposit is sent to me (if broken) or to you (if ok)
  - if we don’t agree a neighbor will act as judge

PART III

truffle & metamask

Truffle is JS Tooling

- Compile solidity
- Manage dependencies
- Deploy contracts into the network
- Unit testing
- Helps to build web apps

Metamask a private key container

- Create/import private keys
- Sign transactions generated by browser
- Transfer ethers between accounts

PART IV

swarm & zk-snarks

Swarm is a content distribution network

- Massive distributed hash table
- Clients accesses data by their hash
- Servers are paid to store those data
- Like IPFS https://ipfs.io/ipfs/QmRqADN7ix21cMVQtKJi6KJP7aFNLvYH1h5BNkQtkn4F56/

zk-snarks are privacy-preserving proofs (homomorphic encryption)

- Cipher(a)+Cipher(b) === Cipher(a+b)