

The title is framed by a large dashed rectangle. At the top-right corner, a dashed arrow points downwards and to the left. At the bottom-left corner, a dashed arrow points upwards and to the right. A solid vertical line with an arrowhead at the bottom is positioned on the right side of the dashed frame.

EOSIO And Smart Contract Development

Hello Virginia Tech!

I am Larry Kittinger

I am a software engineer
on the blockchain team at
block.one.

You can message me at:
larry.kittinger@block.one



1

Getting Started



What you need to build
contracts and run them.



Developer Portal

<https://developers.eos.io/>

We will be using the EOSIO Web IDE for our development concerns.

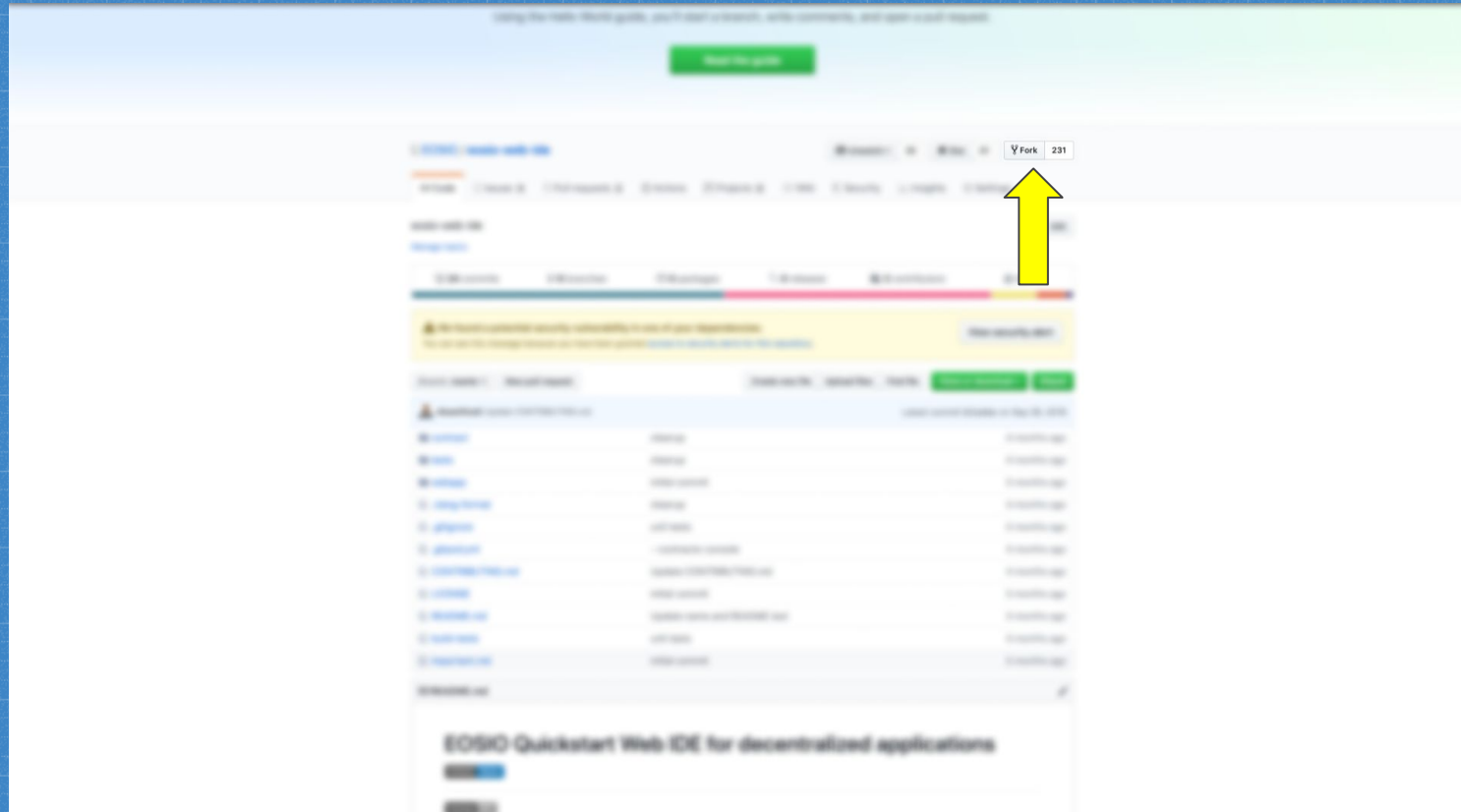
What you need-

- 1) You will need a *github* account, this will be used to fork the web ide.
- 2) Cursory knowledge of C++.
- 3) Basic knowledge of the EOSIO platform.

Getting Started with EOSIO Web IDE

- 1) Goto the url <https://github.com/EOSIO/eosio-web-ide>.
- 2) Fork the repository to your account.
- 3) Navigate to the url:
<https://gitpod.io/#https://github.com/<your account name>/eosio-web-ide>
Fill in your account name above.

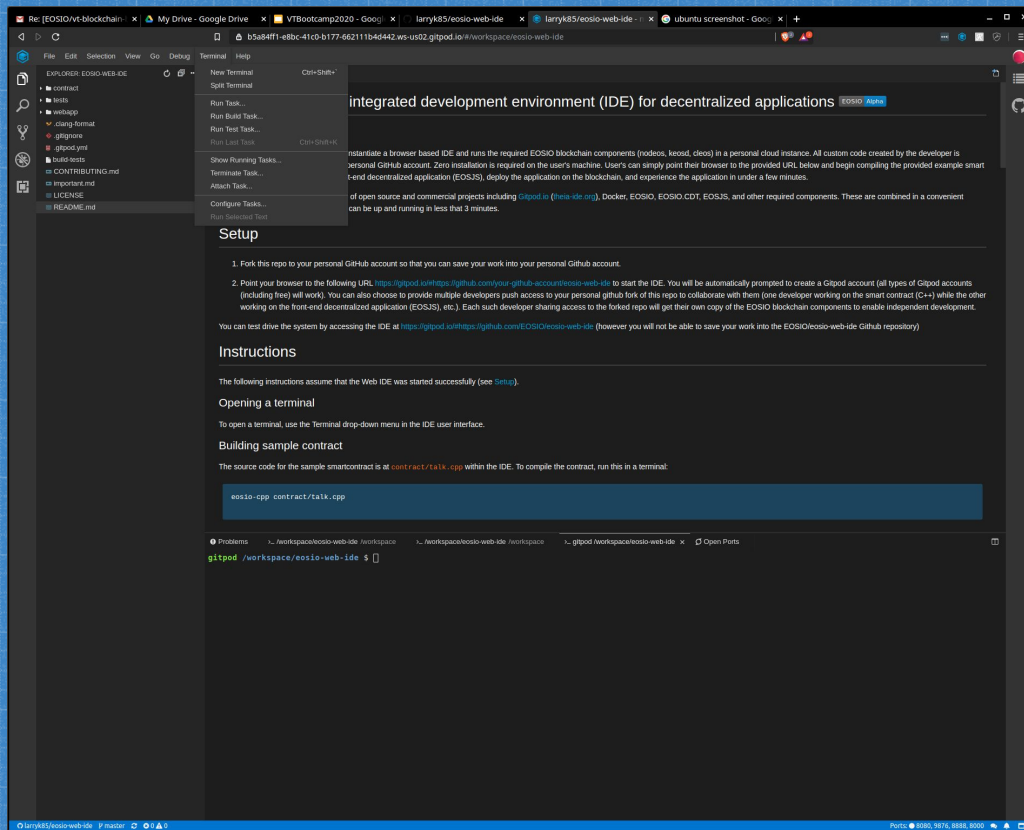
Getting Started with EOSIO Web IDE



Getting Started with EOSIO Web IDE

Once you have logged in. You should see the output of a blockchain already running. From here you will need to open a terminal so you can use `ctrl+shift+`` or navigate to the terminal tab and start a new terminal that way.

Getting Started with EOSIO Web IDE



Getting Started with EOSIO Web IDE

You will need to clone the [vt-blockchain-bootcamp-starter](https://github.com/eosio/vt-blockchain-bootcamp-starter) repository.

So, use the command

```
git clone https://github.com/eosio/vt-blockchain-bootcamp-starter
```

This should clone the starter repo into the home directory.

Getting Started with EOSIO Web IDE

```
gitpod /workspace/eosio-web-ide $ git clone https://github.com/eosio/vt-blockchain-bootcamp-starter
Cloning into 'vt-blockchain-bootcamp-starter'...
remote: Enumerating objects: 117, done.
remote: Counting objects: 100% (117/117), done.
remote: Compressing objects: 100% (78/78), done.
remote: Total 536 (delta 39), reused 102 (delta 26), pack-reused 419
Receiving objects: 100% (536/536), 663.51 KiB | 12.29 MiB/s, done.
Resolving deltas: 100% (310/310), done.
gitpod /workspace/eosio-web-ide $ ls
build-tests  contract  CONTRIBUTING.md  important.md  LICENSE  README.md  tests  vt-blockchain-bootcamp-starter  webapp
gitpod /workspace/eosio-web-ide $
```


Getting Started with EOSIO Web IDE

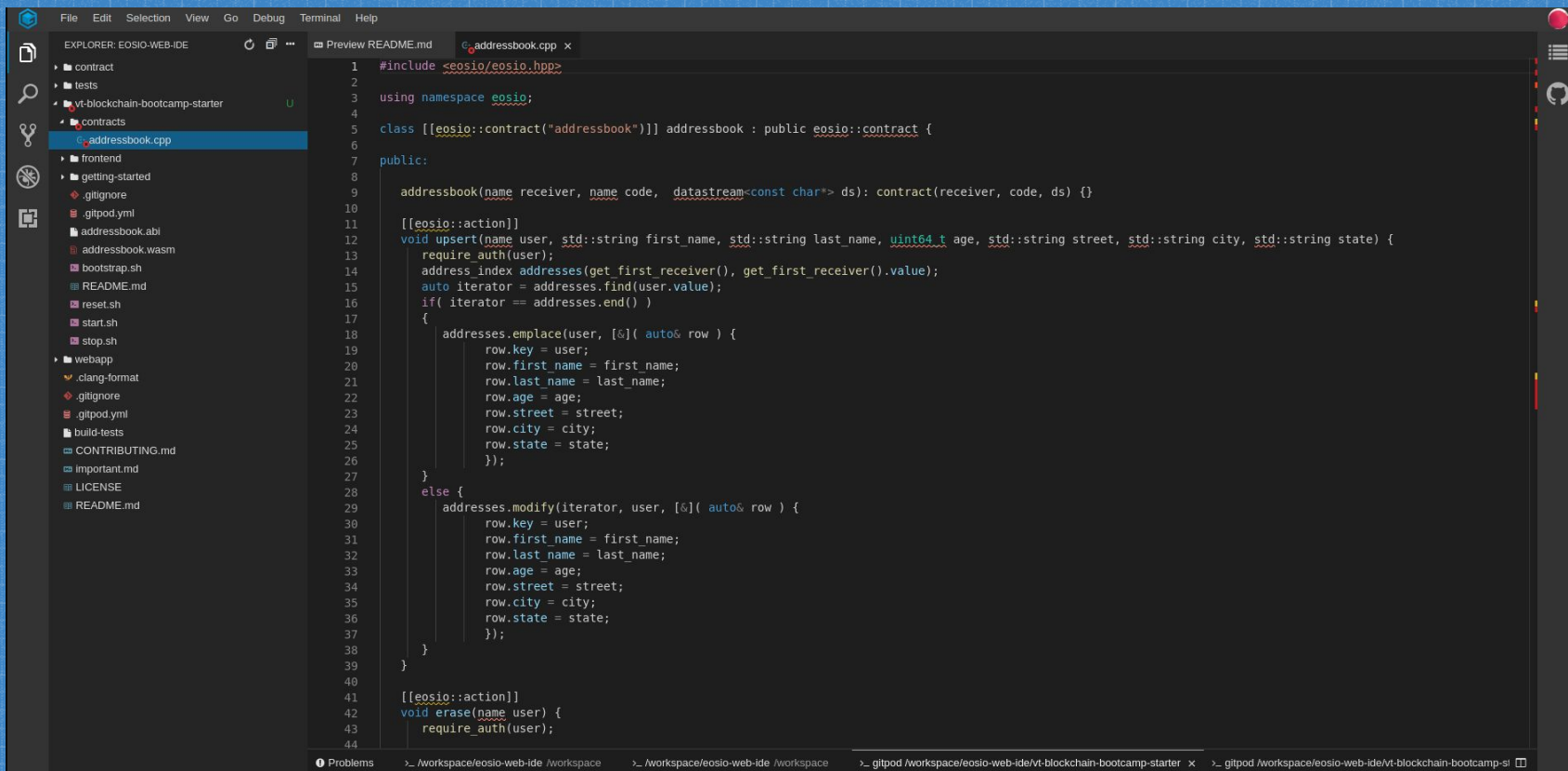
In the `vt-blockchain-bootcamp-starter` repo there is:

- The addressbook C++ source code for the “smart contract”.
- The web frontend Javascript source code.
- A set of scripts to help you quickly start, reset and stop the blockchain.
 - The reset script will stop the blockchain and reset the data stored for the blockchain, so when you restart the chain it is starting from the beginning. You can use this if you get the state in an inconsistent state or “jam” your blockchain.

Getting Started with EOSIO Web IDE

Integrated into this IDE is a VSCode like environment. So you can edit your smart contracts and frontend code in your browser. Some VSCode plugins will work with this setup, it will be a bit of trial and error for the developer to see if their favorite plugins will work or not.

Getting Started with EOSIO Web IDE



The screenshot displays the EOSIO Web IDE interface. On the left is a file explorer showing a project structure with folders like 'contract', 'tests', and 'frontend', and files like 'addressbook.cpp', 'README.md', and 'LICENSE'. The main area is a code editor showing the content of 'addressbook.cpp'. The code is a C++ EOSIO contract named 'addressbook'.

```
1 #include <eosio/eosio.hpp>
2
3 using namespace eosio;
4
5 class [[eosio::contract("addressbook")]] addressbook : public eosio::contract {
6
7 public:
8
9     addressbook(name receiver, name code, datastream<const char*> ds): contract(receiver, code, ds) {}
10
11     [[eosio::action]]
12     void upsert(name user, std::string first_name, std::string last_name, uint64_t age, std::string street, std::string city, std::string state) {
13         require_auth(user);
14         address_index addresses(get_first_receiver(), get_first_receiver().value);
15         auto iterator = addresses.find(user.value);
16         if (iterator == addresses.end()) {
17             {
18                 addresses.emplace(user, [&]( auto& row ) {
19                     row.key = user;
20                     row.first_name = first_name;
21                     row.last_name = last_name;
22                     row.age = age;
23                     row.street = street;
24                     row.city = city;
25                     row.state = state;
26                 });
27             }
28         }
29         else {
30             addresses.modify(iterator, user, [&]( auto& row ) {
31                 row.key = user;
32                 row.first_name = first_name;
33                 row.last_name = last_name;
34                 row.age = age;
35                 row.street = street;
36                 row.city = city;
37                 row.state = state;
38             });
39         }
40     }
41
42     [[eosio::action]]
43     void erase(name user) {
44         require_auth(user);
```

The bottom status bar shows the current file path: `~/workspace/eosio-web-ide /workspace`.



2

EOSIO Platform



What is EOSIO.



What is EOSIO?

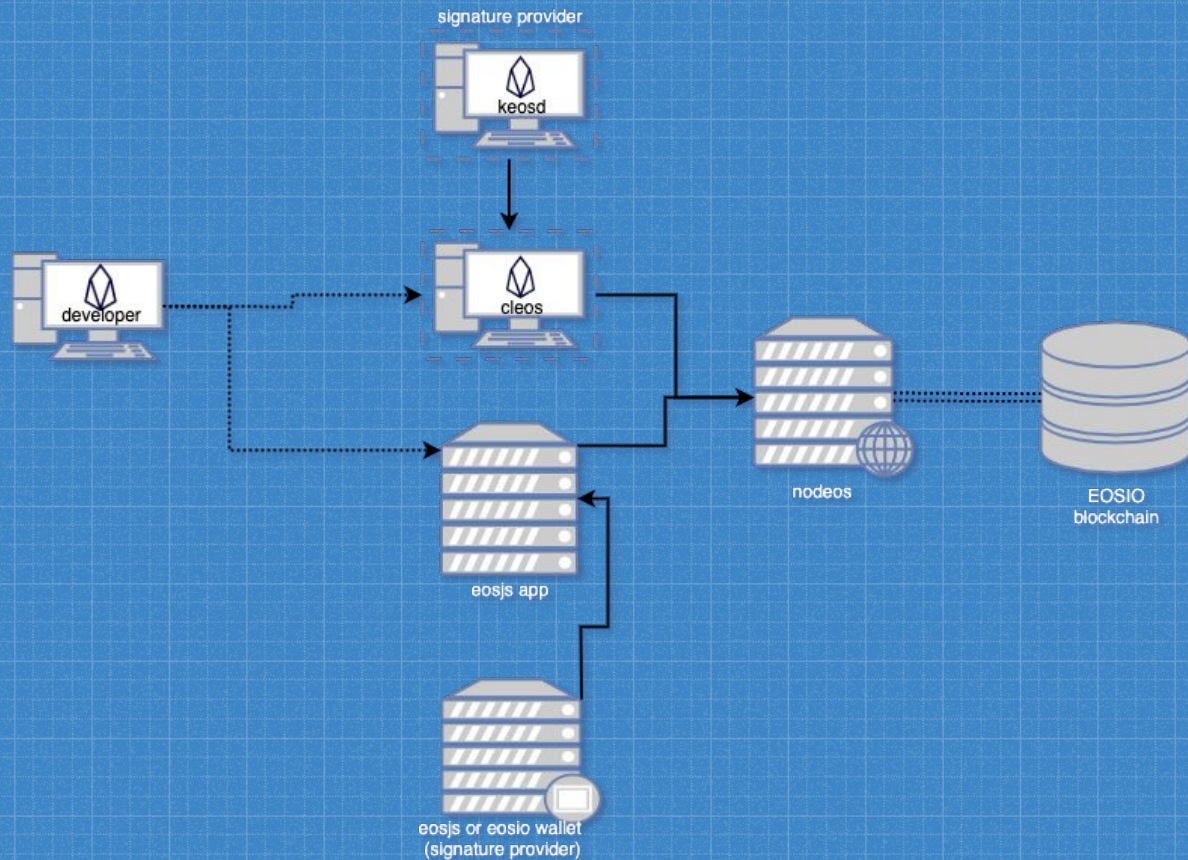
EOSIO is a free, open-source blockchain software protocol that provides developers and entrepreneurs with a platform on which to build, deploy and run high-performing blockchain applications.

What is EOSIO?

The suite of EOSIO programs:

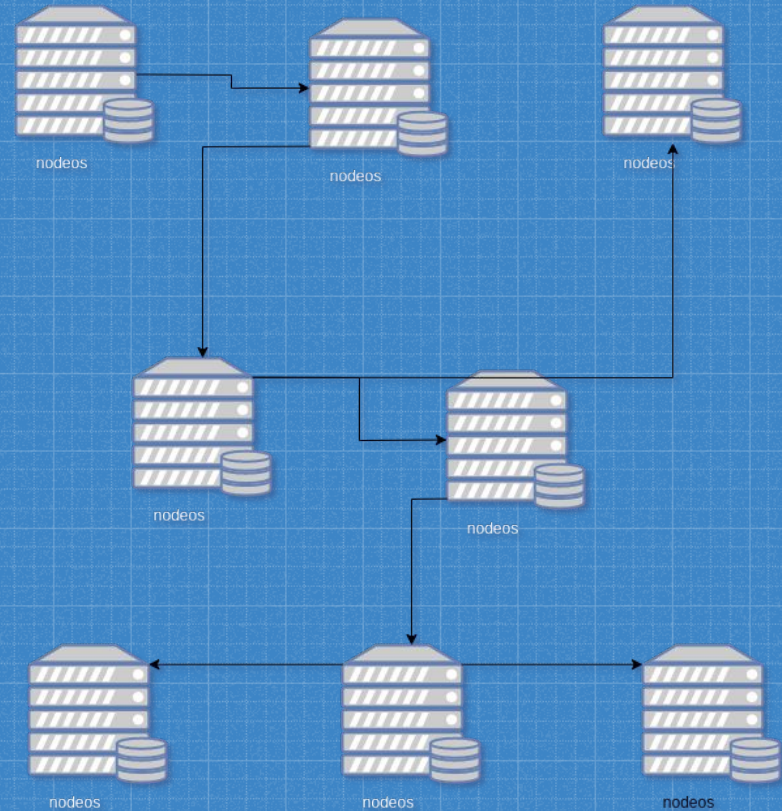
- ***nodeos***
 - This is the program that is the blockchain node (so the blockchain effectively). This exposes an API to end users and is called the NODEOS RPC API. This is used to interact with the blockchain and push actions or query data.
- ***cleos***
 - This is the client tool to communicate with the running ***nodeos*** instance. This is how you will push actions and query data from the blockchain.
- ***keosd***
 - This is the signature provider daemon that will sign transactions via the ***cleos*** tool.

What is EOSIO?



What is EOSIO?

Nodeos is can be a single instance or a network of nodes. These nodes can communicate with each other through a p2p network.



What is EOSIO?

As a developer you will either be interacting with the tool **cleos** directly or with an **eosjs** frontend. Both of these will interact with the **nodeos** through the RPC API.

Jeffrey Smith will discuss the Javascript interface to this API and I will quickly go over the main commands that you will use with **cleos** that interact with the RPC API.

What is EOSIO?

nodeos

As mentioned before exposes the RPC API that you will interact with during your development. This also handles all of the forking logic, consensus, WASM virtual machines and a database of for blockchain state.

What is EOSIO?

cleos

This will be your main entry point into the *eosio* system during your development.

Some of the common commands that you will need are:

- `cleos get info`
 - This will return back the current head block and display information about it.
- `cleos set code <account> <path-to-contract>/<contract>.wasm`
 - This will set the smart contract in question to the given account.
- `cleos set abi <account> <path-to-abi>/<contract>.abi`
 - This will set the ABI for the smart contract to the given account.
- `cleos create account <creator> <account name> <public key>`
 - This will create an account on the blockchain. This will need a creator to “endorse” the creation, for most of your work using the account *eosio* is perfectly fine.

What is EOSIO?

cleos

This will be your main entry point into the *eosio* system during your development.

Some of the common commands that you will need are:

- `cleos create key`
 - This will create a new public key and private key pair to use when creating a new account on your chain.
- `cleos wallet import --private-key <priv-key>`
 - This will import the private key (priv-key) into your wallet to allow for signing transactions later.
- `cleos push action <account> <action> ' [<parameters>]' -p <authorizer>`
 - This will push an action against the account specified with the parameters in JSON form given the authority of the authorizer.

What is EOSIO?

What is an authority?

- An authority is a public key associated with an account and a permission level.
- A permission level is a quite complex thing to go into for this talk (see [Accounts and Permissions](#)), but by default two of these levels are generated for all new accounts.
 - **owner** - this acts as your ultimate authority and can satisfy any type of authorization needed by your account.
 - **active** - this one is your daily driver permission and will satisfy most types of authorizations needed, but this can be reset from your owner permission if lost or stolen.

What is EOSIO?

What is an action?

There are two definitions based on context:

- An action is a function in a Webassembly binary that will be invoked with a *push action* command and invoked with a set of parameters given to it as a payload of data. This definition is used when talking about actions when developing the smart contract.
- An action is an eosio **name** for the account this action is going to be run against, an eosio **name** for the action name itself and a payload of data to act on. This is the definition used when looking at transactions and when interfacing with cleos or EOSJS.

What is EOSIO?

What is a transaction?

A transaction is a set of actions. Possibly one action, and for most of the cleos commands this will be only one, but N actions can comprise a transaction. The transaction also contains the set of authorizations that authorized it.

What is EOSIO?

What is an eosio name?

An eosio name is a string that is encoded into a 64 bit variable. As such it has some none obvious restrictions on them. They can be 12 characters long, and a character can only be one of [abcdefghijklmnopqrstuvwxyz.12345]. These are used for account names, permission names, action names, table names, notification handler names, and secondary index names.

So,

eosio, name, bob, alice, bob15, b.ob34 are all valid eosio names

But,

Eosio, nAme, 0bob, alicealicealic are invalid eosio names

What is EOSIO?

What is an eosio symbol?

An eosio symbol is an all uppercase string of 7 or less characters from [A-Z] and a precision.

The precision is used to determine how many decimal places are there, this is a fixed point system.

We will see this in more detail when we talk about eosio.token.

What is EOSIO?

What is an eosio intrinsic?

Given that the smart contracts are compiled to WebAssembly, creating side effects outside of the WASM sandbox needs to be added. These are what we refer to as eosio intrinsics, they act much like syscalls in OS parlance. These are things like printing, asserting, table operations. Some of these intrinsics are there because of performance concerns, so things like cryptographic hashing is done through these intrinsics.

What is EOSIO?

What is an eosio table?

These are persistent data storage used by smart contracts. These are defined as either C++ structs or classes. Intrinsic are available to allow for basic database operations to act on these tables, so, CRUD operations. These intrinsic also allow for iteration through the tables and allow the developer to walk through tables in a manner much like C++ iterators.

What is EOSIO?

What is WebAssembly (WASM)?

WebAssembly is a platform agnostic assembly language that emphasizes determinism, simplicity and security. **Nodeos** uses WASM as the smart contract language that is executed. The developer does not need to implement smart contracts in WebAssembly, a toolchain is provided to compile smart contract written in C++ to this binary format.

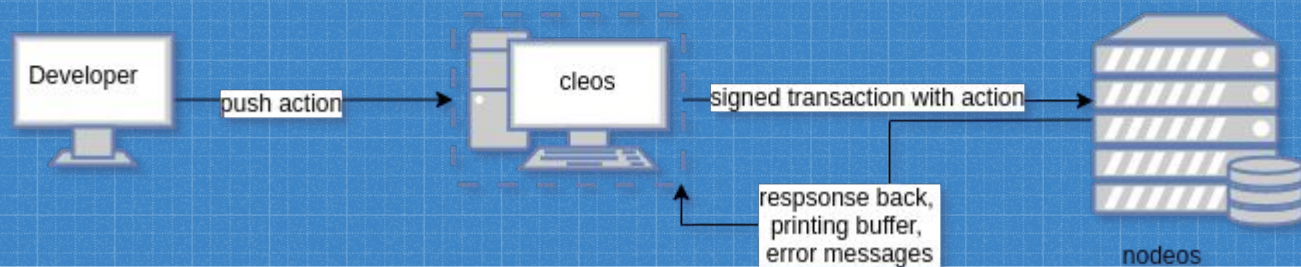
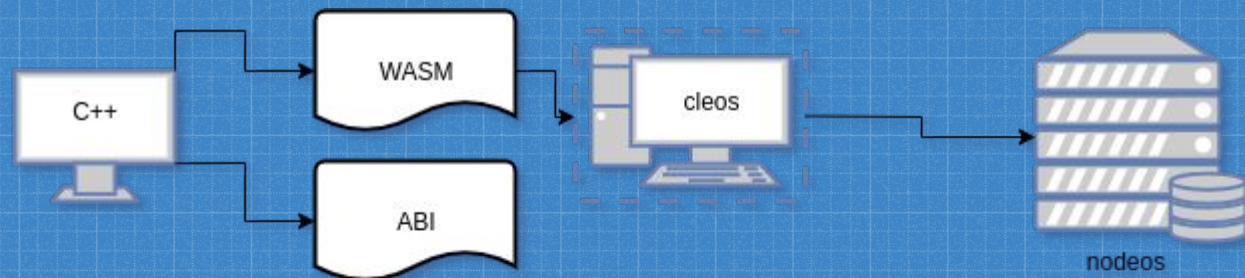
What is EOSIO?

What is an ABI?

This is a data layout and conversion schema for your smart contract. This will allow tools to interact with your smart contracts in an intelligent way and pack and unpack data needed by an action or from events.

This is used by cleos to allow for JSON inputs for parameters and generate the correct binary data needed to satisfy your action and interpret table structures when querying. This is also used by EOSJS to do the same type of things but with the control and expressiveness of Javascript.

What is EOSIO?



What are smart contracts?

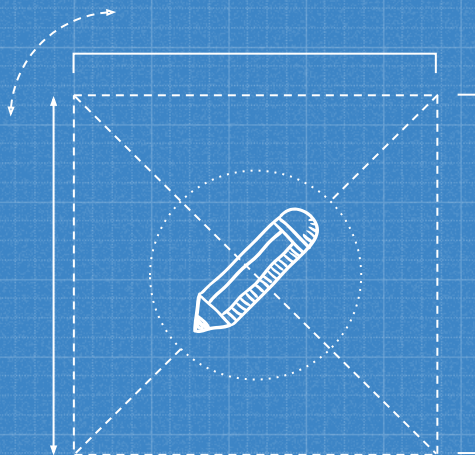
Smart contracts are a blob of code that digitally facilitate, verify or enforce a contract between N members.

This is general enough of an definition that allows for a great deal of things to fit into this model.

What are smart contracts?

This can include:

- Escrows
- Voting
- Supply chain management
- Work time management
- Real estate brokerage
- Healthcare records
- Gaming
- Social Media



Hello, World!

Hello, World!

```
#include <eosio/eosio.hpp>

class [[eosio::contract]] hello : public eosio::contract {
    public:
        using eosio::contract::contract;

        [[eosio::action]]
        void hi() {
            eosio::printf("Hello, World\n");
        }
};
```


Hello, World!

We need to define this class as an eosio contract. This requires that you inherit from the `eosio::contract` class (this is found in `contract.hpp`). You will also need to add the attribute `[[eosio::contract]]` to direct the compiler to generate the main dispatcher and for **ABI** generation.

```
class [[eosio::contract]] hello : public eosio::contract {
```

Note that you can explicitly name this attribute.

```
class [[eosio::contract("hello_contract")]] hello : public eosio::contract {
```

The first example will use the name of the class itself for the attribute name.

The attribute name is ultimately used to logical connect actions, tables and notification handlers to a specific compiled smart contract.

Hello, World!

This line is used to tell the compiler to use the constructors of the base class.

```
using eosio::contract::contract;
```

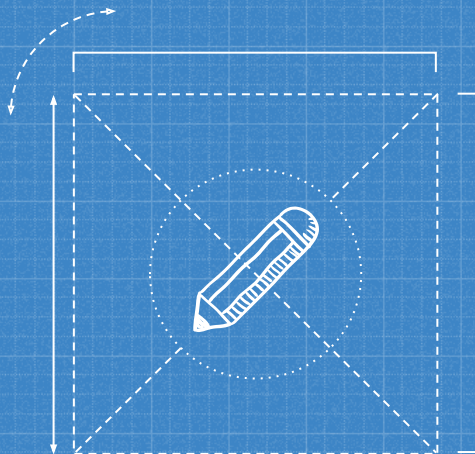
You need to mark a method as an `eosio::action`. This will be an entry point into your smart contract (more on this in the next slide).

```
[[eosio::action]]  
void hi() {  
    eosio::printf("Hello, World\n");  
}  
};
```

This can be invoked with `cleos push action <account> hi '[]' -p <account>`.

Hello, World!

When you push the **hi** action against the hello world contract. Nodeos will find the account that you are pushing against and load the deployed WASM into a sandboxed VM. From there it will call the main entry point of the smart contract, this is a function called ***apply***. The compiler will generate the code to instantiate your smart contract class with the parameters that it needs and call the method that is associated with the action that you are invoking with the unpacked data from the input payload.



eosiolib

The library that will be your life for the next few months.

eosiolib

contract.hpp

This as mentioned before is the class that describes an eosio smart contract “container”. The compiler will generate the constructor for this class for you and dispatch/call the method that is associated with the action you want to call. The internal variables, the “self” name of the account this contract is deployed to, the “first receiver” name of the account this action first came from and the datastream object that is a pointer and stream object over the data blob that was passed in.

eosiolib

name.hpp

This is the class that represents a valid **eosio** name. This will take a string for the constructor and at compile time (if it can) generate the conversion to the `uint64_t` representation.

You can construct an `eosio::name` like so ...

```
eosio::name account("bob");
```

You can also use the user defined suffix (`_n`) for this ...

```
auto account = "bob"_n;
```


eosiolib

symbol.hpp

This class represents an **eosio** symbol. This type can take a string as an input and an explicit precision.

So, you would use this like this ...

```
eosio::symbol usd("USD", 2); // this could represent a US $  
eosio::symbol vt("VT", 5);
```


eosiolib

asset.hpp

This class represents a symbol with an amount. So this would be used to create “tokens” and the like, and ensure that calculations and modifications on the asset are done with either scalar values or the same type of symbol.

So,

```
eosio::symbol usd("USD", 2);  
eosio::asset my_money(40, usd);  
eosio::asset your_money(1000, {"VT", 5});  
my_money += your_money; // this should fail
```


eosiolib

check.hpp

The `eosio::check` functions allow you to check whether a predicate is true and assert and halt if it is false. These are given the predicate and an assertion message if the failure occurs.

So,

```
eosio::check(true, "should never fail");  
bool definitely_true = false;  
eosio::check(definitely_true, "something bad happened");
```


eosiolib

multi_index.hpp

This is our abstraction over the base table intrinsics and allow the developer to create these table objects and define primary and secondary keys to access data from them. Some more detail on these when we look at the addressbook example and some description of how eosio tables are used in general.

eosiolib

action.hpp

This houses our abstractions for an eosio action. These are either constructing an `eosio::action` object from all of the parts that you need and then calling the intrinsic `send` on them. Or the preferred method of using action wrappers to send these actions.

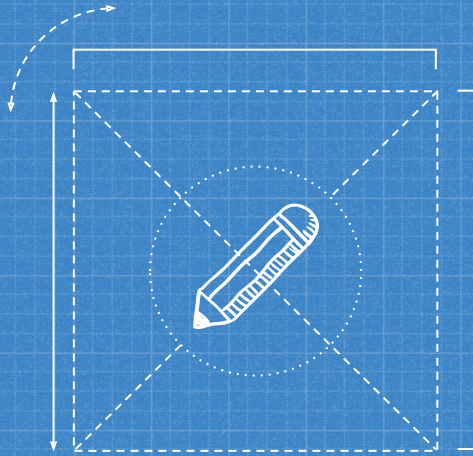
Something like this,

```
using transfer_act = eosio::action_wrapper<"transfer"_n,  
&token::transfer>;  
transfer_act transfer("token"_n, {get_self(), "active"_n});  
transfer.send("bob"_n, "alice"_n, 400);
```


`eosiolib`

`everything else.hpp`

There are quite a few more types and functions that you can use for your smart contract development. These are things like time, system level functionality, permissions,



Addressbook

A smart contract that acts as an address book and showcases actions and tables

addressbook

We are going to be using a lot of eosio namespace a lot, so let's add this using

```
using namespace eosio;
```

Note that we explicitly "named" this contract to addressbook and flagged to the compiler that this will be our smart contract container.

```
class [[eosio::contract("addressbook")]] addressbook : public eosio::contract {
```

```
...
```

Note this constructor, because we have to initialize local members we need to make an explicit constructor and not simply use the parent classes. The initialization of the member addresses (the table) needs an owner of that table and a scope. Don't concern yourself with what a scope is, simply use this `get_self().value` as a pattern. And for your work making the owner of the table as `get_self()` is perfectly valid to do.

```
    addressbook(name receiver, name code, datastream<const char*> ds):  
        addresses(get_self(), get_self().value),  
        contract(receiver, code, ds) {}
```


addressbook

```
[[eosio::action]]  
    void upsert(name user, std::string first_name, std::string last_name,  
                uint64_t age, std::string street, std::string city,  
                std::string state) {
```

```
[[eosio::action]]  
    void erase(name user) {
```

```
[[eosio::action]]  
    void print(name user) {
```

```
[[eosio::action]]  
    void ge(uint64_t age) {
```

Note that these actions can take arbitrary input and there is no restriction on this besides C++'s restrictions on max function parameters.

addressbook

This is defining our “table” entry that we will use to create our multi_index data type from.

```
struct [[eosio::table]] person {  
    name key; // Note that we need some sort of key value  
    std::string first_name; // these can be any type of value that you'd like  
    std::string last_name;  
    uint64_t age; // since we are going to define a secondary index for this  
    std::string street;  
    std::string city;  
    std::string state;  
    uint64_t primary_key() const { return key.value; } // needed by multi_index  
    uint64_t get_age() const { return age; } // needed to index by your secondary  
};
```


addressbook

This defines our multi_index table object type. We need to name it with an eosio name so that the ABI can be generated for this and external tools can reference it (like cleos or EOSJS). We then need to give it the type of the table entry that we want to use. The next section is only needed if you have secondary indices on your table. Since we are we need to create this indexed_by type that takes an eosio name to reference it externally and a const_mem_fun type. The const_mem_fun type, is a wrapper over a constant member function (method), so we need to tell it what table entry type again, the type of this secondary key and the member function pointer that we want to use.

```
using address_index = eosio::multi_index<"people"_n, person,  
    indexed_by<"byage"_n, const_mem_fun<person, uint64_t, &person::get_age>>>;
```


addressbook

This will require that the authorizer of the transaction was in this case
`user`
`require_auth(user);`

From our constructed `multi_index` type we want to find the user. Since the primary key has to be a `uint64_t`, we need to use the underlying value of the name.

```
auto iterator = addresses.find(user.value);
```

Much like normal C++ iterators we can check if the iterator is pointing to the `end()` and know that the element is not in the table.

```
if( iterator == addresses.end() )  
{
```

```
    addresses.emplace(user, [&]( auto& row ) {  
        row.key = user;  
        row.first_name = first_name;  
        row.last_name = last_name;
```


addressbook

Since the element was not found we need to `emplace` one into the table. For this we call the `emplace` method on the `multi_index` object with the payer of the table, since most of your work will be without resources this can be the user or `get_self()`, or any account on the blockchain. The next parameter is the “constructor”, this is a closure or lambda that will let you modify the element. Please note that `auto` and `auto&` are not the same thing.

```
addresses.emplace(user, [&]( auto& row ) {  
    row.key = user;  
    row.first_name = first_name;  
    row.last_name = last_name;  
    row.age = age;  
    row.street = street;  
    row.city = city;  
    row.state = state;  
});
```


addressbook

Since the element already exists in the table, we will only update the object. The `modify` method will take an iterator to the element in question, the payer of the row and a closure or lambda that takes as input an *auto&* reference.

```
addresses.modify(iterator, user, [&]( auto& row ) {  
    row.first_name = first_name;  
    row.last_name = last_name;  
    row.age = age;  
    row.street = street;  
    row.city = city;  
    row.state = state;  
});
```


addressbook

We are going to get the secondary index called *byage*. This index can be used in much the same way as the base `multi_index` object, including using `upper_bound` and `lower_bound` and iterating.

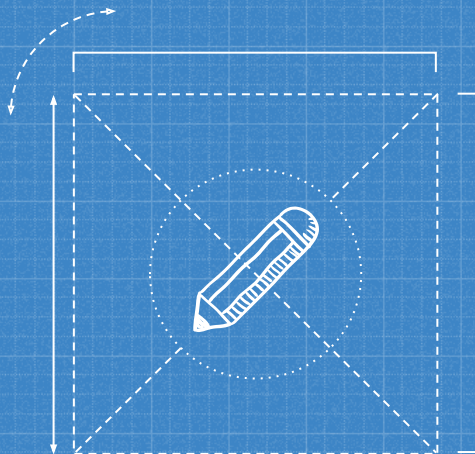
```
const auto& byage_index = addresses.get_index<"byage"_n>();  
for (auto itr = byage_index.lower_bound(age); itr != byage_index.end(); ++itr) {  
    print_method(*itr);  
}
```


addressbook

For your cleos commands always use the `-v` option.

And to add the authority of code to your account to send inline actions use:

```
cleos set account permission <account> active --add-code
```

eosio.token

The crypto currency template/generator for the eosio platform.

`eosio.token`

This defines the eosio token “standard”. The pattern of the input parameters to the transfer action, the table layout and some of the other conventions allow other developers and users of the blockchain to define (create and issue) their own token types and they will be viewable by external block explorers and other tooling.

`eosio.token`

The cryptocurrency of the eosio platform can fundamentally be anything that you would like it to be. The `eosio.token` is a reference implementation and is one way to handle tokens. But, if you wanted a more simplistic system then one could create a table that keeps track of balances for users and uses a basic `int64_t` or `int32_t` type to be the balances themselves. The concept of cryptocurrencies at the highest level on the eosio platform is completely virtual and redefinable as the needs of the particular running network.

Thanks!

ANY QUESTIONS?

You can email me at:

`larry.kittinger@block.one`



SlidesCarnival icons are editable shapes.

This means that you can:

- Resize them without losing quality.
- Change fill color and opacity.

Isn't that nice? :)

Examples:

