.NET Conf CO

v2018

Working with Azure Blockchain and .NET Core

Speaker Information

Blockchain

Introducción

Blockchain

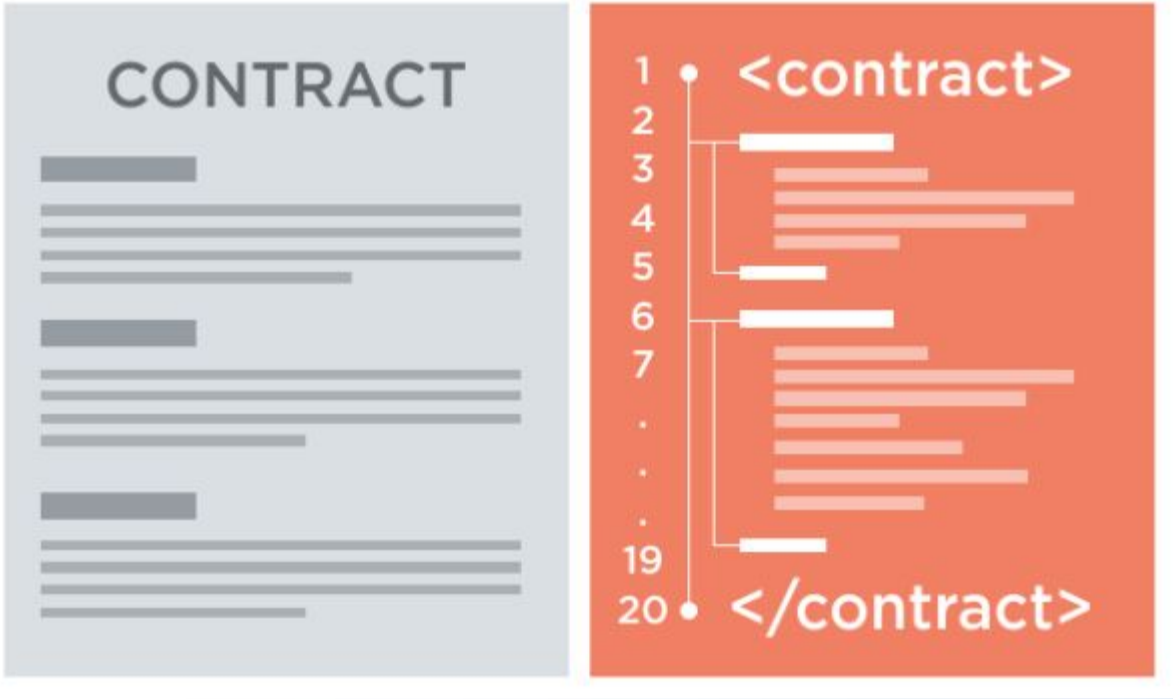
- Transparencia
- Inmutabilidad
- Bajo Costo
- Transacciones rápidas
- Seguridad

Bitcoin

- Transparencia
- Inmutabilidad
- Bajo Costo
- Transacciones rápidas
- Seguridad

Contratos Inteligentes

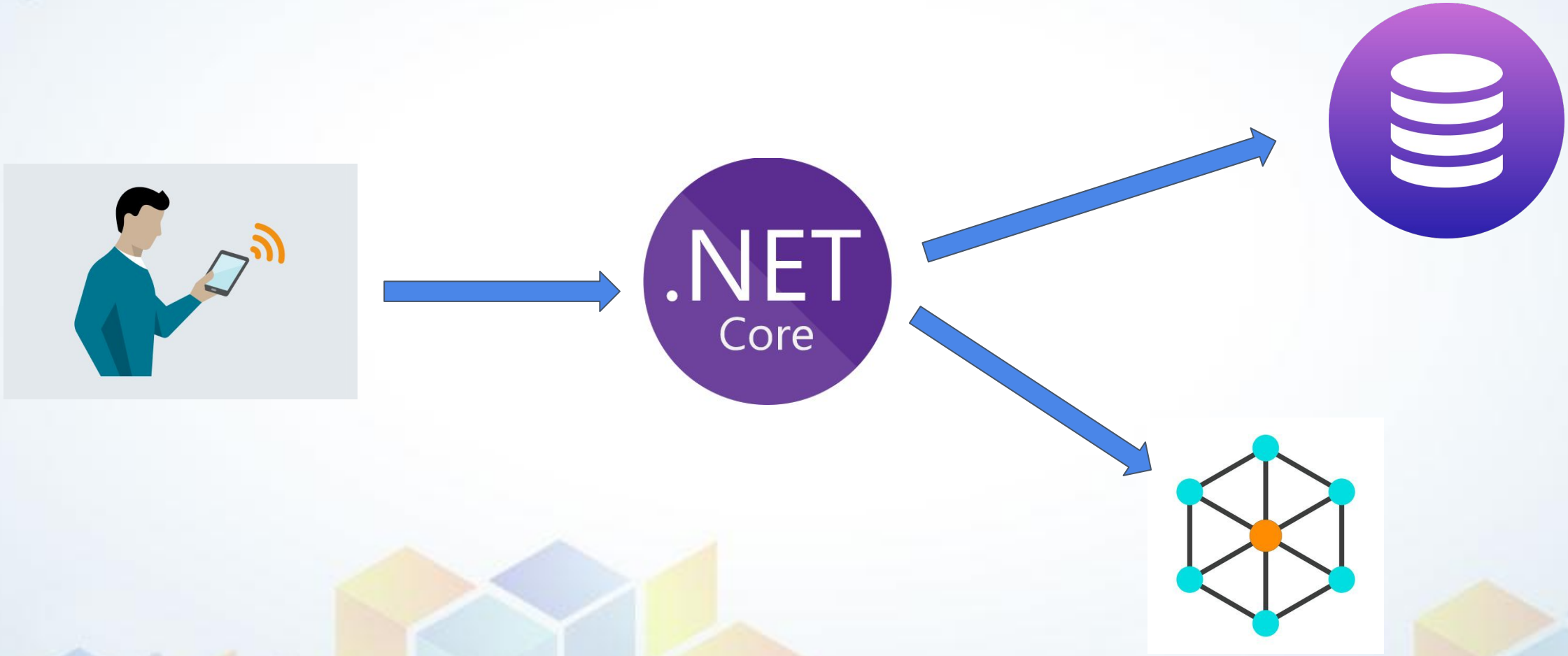
- Transparencia
- Inmutabilidad
- Bajo Costo
- Transacciones rápidas
- Seguridad
- Auto – Verificable
- Auto – Ejecutable
- Conciliación Automática



Preparando el entorno

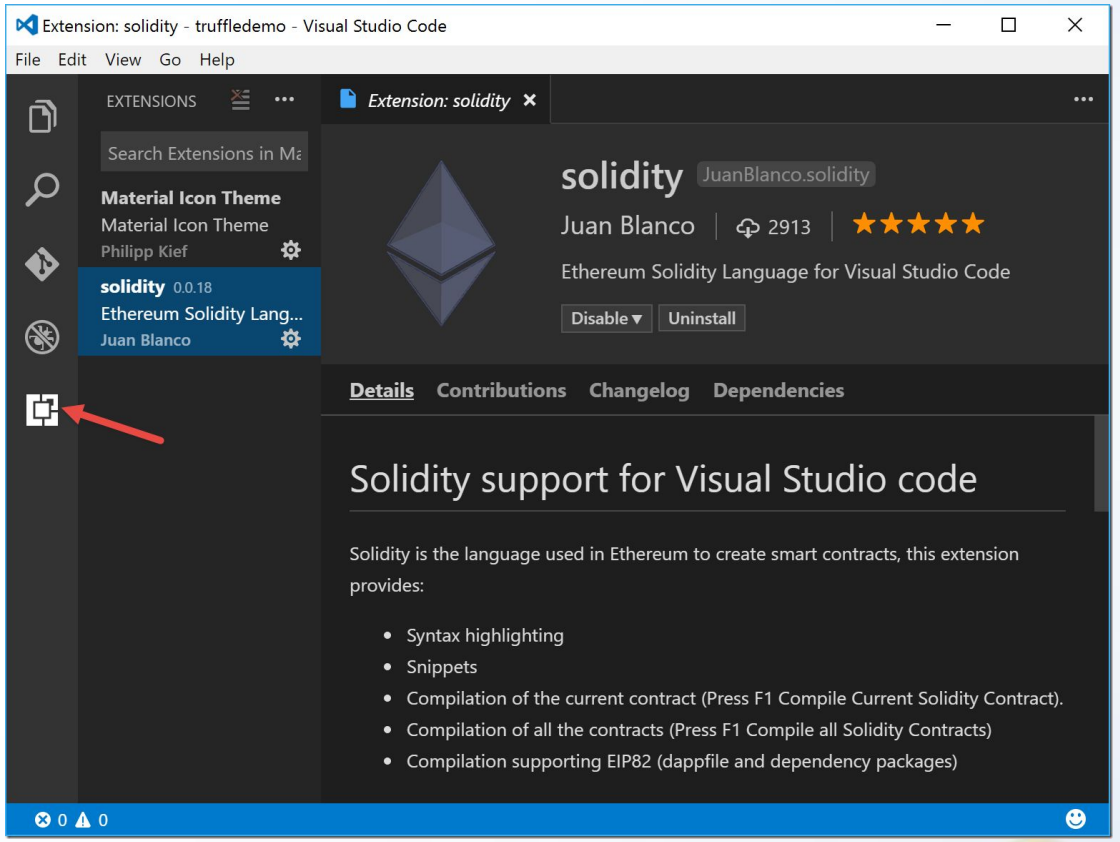
Desarrollando Smart Contracts

Arquitectura de una Dapp



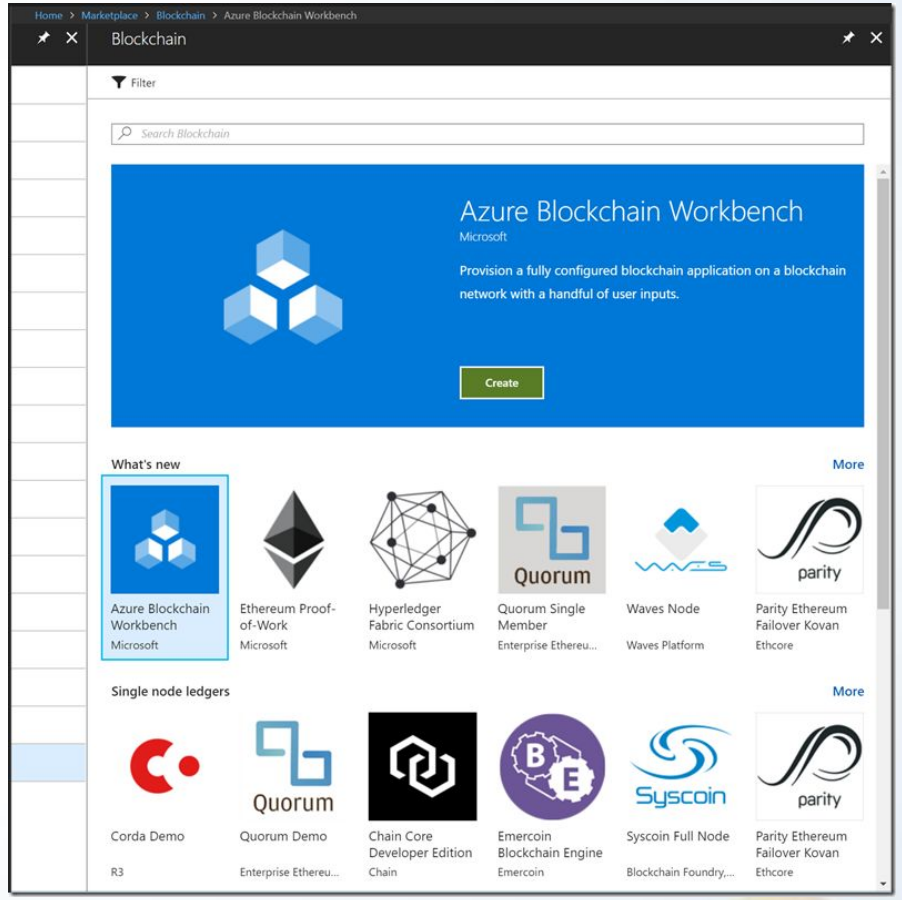
Entorno de Desarrollo

- Visual Studio Code
- Solidity Plugin (Juan Blanco)
- .NET Core
- Nethereum



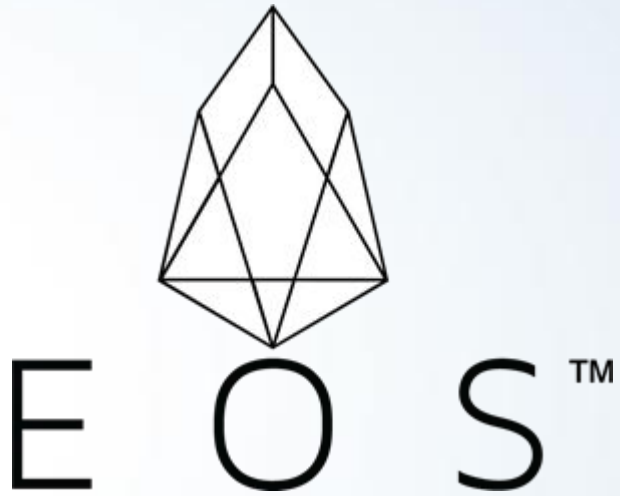
Infraestructura

- Nodos Mainnet, Testnet, Regtest
- Azure Marketplace > Blockchain
- Azure Blockchain Workbench



Infraestructura

- Ethereum
- EOS
- RSK



RSK

- EVM 100% compatible con Ethereum
- Alta seguridad
- Contratos costeados en Bitcoin
- http://rsk.co

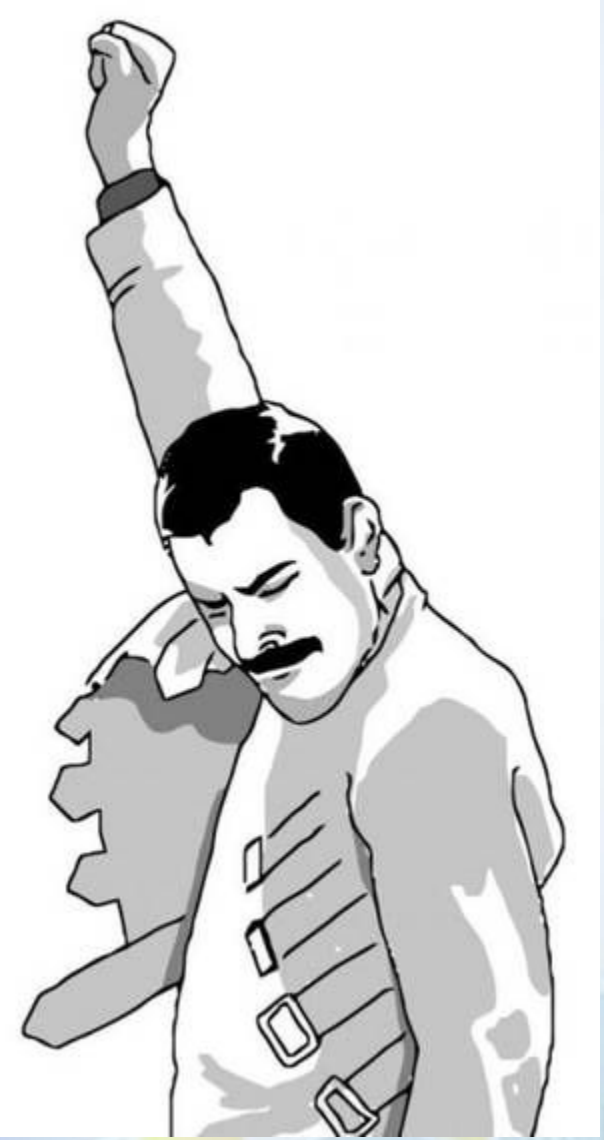
Crear un nodo de RSK en Azure

Crear un nodo local - regtest

- Instalar Java JDK 8
- Entrar a https://github.com/rsksmart/rskj/releases
- Bajar la última versión (0.5.3 - ORCHID)
- Abrir una terminal en la carpeta donde bajamos el jar
- Correr el siguiente comando:
  - `java -cp ./rskj-core-0.5.3-ORCHID-all.jar co.rsk.Start --regtest`

Corroborar que todo esté bien

- Bajar el proyecto RSK Manager de mi github
  - http://gihub.com/garudaslap
- Abrirlo con Visual Studio
- Ejecutar el test “GetAccountsTest”




.NET Conf CO
v2018
Let’s work !
Manos a la obra !

Crearemos una nueva cuenta

- Corroboraremos que existe la cuenta “Cow” con el Test “CheckIfCowAccountExists”
- Corroboraremos si tiene saldo con “CheckBalanceOfCowAccount”
- Llamaremos al método “CreateNewAccount” poniendo un breakpoint en la última linea del Test para ver la información de la cuenta
- Guardaremos el address y la clave privada en los settings
- Corroboraremos el saldo 0 de la nueva cuenta con “CheckBalanceOfCreatedAccount”

Crearemos una nueva cuenta

- Corroboraremos que existe la cuenta “Cow” con el Test “CheckIfCowAccountExists”
- Corroboraremos si tiene saldo con “CheckBalanceOfCowAccount”
- Llamaremos al método “CreateNewAccount” poniendo un breakpoint en la última linea del Test para ver la información de la cuenta
- Guardaremos el address y la clave privada en los settings
- Corroboraremos el saldo 0 de la nueva cuenta con “CheckBalanceOfCreatedAccount”

Daremos saldo a la cuenta

- Enviaremos saldo a la cuenta con el método “SendGasFromCow…”
- Corroboraremos si tiene saldo ahora con “CheckNewBalance…”

Crearemos un contrato

- Abriremos el contrato HelloWorld del repositorio con Visual Studio Code
- Presionamos “F1” y buscaremos “compile current contract”
- Aparecerá en la carpeta “bin” la compilación, de ahí sacaremos el bytecode y el abi para guardarlos en los settings
- Llamaremos al método “DeployContract” del Test y con un breakpoint en la salida chequearemos el address del mismo para guardarlo en los settings
- Probaremos llamar a “CallContract” para ver que nuestro contrato se implementó correctamente

Cambiaremos el estado

- Le daremos nuevo saldo a nuestra cuenta
- Crearemos un nuevo Test que llame a la api “callContractFunctionTxParameter” (especial atención a los parámetros)
- Volveremos a llamar al método “callContract” del Test para ver si el saludo se actualizó correctamente


.NET Conf CO
v2018
Felicitaciones !
aplausos !

Sorteo !

Sólo si se cumplen los requisitos :)


.NET Conf CO
v2018
Let’s imagine !
A construir tu propio contrato !

Muchas gracias ! @sebaleoperez

Microsoft

Inter Grupo

ruta n
MEDELLÍN
CENTRO DE INNOVACIÓN Y NEGOCIOS

endava

NAREIA

Fundación AVANET

UruIT

Ceiba software

MASGLOBAL CONSULTING

GROWTH ACCELERATION PARTNERS

Soluciones Móviles areamovil

INTERFAZ PODCAST

DA7OS

intelligent training

TEAM INTERNATIONAL
Innovative Software Solutions

iT Synergy
Growing innovation

.NET Conf CO
v2018