Quorum Consortium Network in Azure Marketplace

Cale Teeter | Senior SDE DX / TED

Overview

The next phase of support and integration of the Quorum blockchain solution in Microsoft Azure is the release of the Quorum Consortium Network in the Azure Marketplace. The goal of this offering is make it easy for user to create the infrastructure and configuration in order to run Quorum by supplying a few parameters and single client for deployment. While the initial Quorum demo offering was a good way to showcase the fundamentals of how Quorum functions and is different from others in the market, this new offering allows users to build and deploy a full network with multiple underlying virtual machines, virtual network, load balancing, and network security, and simple management via the native Cakeshop tooling.

After reading this article, you will

- Understand what the new offering of Quorum Consortium Network architecture is comprised of and how it can be deployed by simply providing a few parameters.
- Learn how to deploy the Quorum Consortium Network and activities to do post deployment to get started with using this offering.

About Quorum blockchain

While there are various types of blockchains on the market today, and more being created every day, Quorum is unique in a few different ways. To start from the core, Quorum is fundamentally, a fork of the go Ethereum client. This means that it is in fact a variant of the Ethereum protocol. This was created by a team at JP Morgan Chase, with the intention of providing a blockchain that can be used by enterprise, addressing some the features that were needed beyond what Ethereum base protocol could provide. Specifically, the modifications were to address the following:

- Need for a different consensus algorithm and model to avoid the challenges that mining presents in a semi-trusted private Ethereum network.
- Need for selective privacy of transactions to allow multiple parties to use a single blockchain, but disclose only transactions details to specific participants.

Based on feedback from enterprises and in fact requirements that JP Morgan Chase was addressing for their own use, these new features were added via a modified go-ethereum client and the use of constellation, or private transaction managers. These additions are built to be the least intrusive approach to implement, meaning the team did set out to reinvent a new blockchain, and instead modified the existing technologies. The specific implementation is the following:

- QuorumChain – a new consensus model based on majority voting, this is also raft-based to ensure faster blocktimes, transaction finality, and on-demand block creation
- Constellation – a peer to peer encrypted messaged exchange
- Peer Security – node/peer permissioning using smart contracts

Architecture

To get a bit deeper into the architecture of how Quorum functions as well as how this deployed in an optimal way to Azure. As mentioned above Quorum is an Ethereum based blockchain with some specific new features.

Consensus

While the ultimate goal is pluggable consensus models, the current implementation is a model named QuorumChain, which is a time-based, majority voting algorithm. This is comprised of a few different pieces.

- Smart contract based model to govern consensus and manage the nodes that can participate as voters.
- Ethereum transactions to propagate votes through the network.
- Ethereum signature validation to validate signatures received from Maker and Voter nodes

The nodes in the Quorum network, can be designated as a Voter role, which allows them to vote on which block should be the canonical head at a particular height. The most recent block with the most votes is considered the canonical head of the blockchain. A block is considered valid once is has enough votes to overcome the threshold defined for votes.

Blocks are created only by nodes with the role of Maker role. Nodes that hold this role can create blocks and sign them by setting their signature in the ExtraData field on the block. When the block in imported to the network/chain, they signature is validated by other nodes to ensure the signer's address in present in the Marker list (whitelist) in the voting contract.

To delve a bit deeper the smart contract to govern the Voters and Makers is named BlockVoting and is deployed to address 0x0000000000000000000000000000000000000020 within the genesis block. The contract is hardcoded into the Quorum client via the address and the ABI for the contract. This means if the consensus rules for voting needs to change, this Quorum client will need to be updated in concert with the smart contract that drives this.

There exist functions in this voting contract to add/remove Voter and Maker nodes as well as the threshold for how many votes are needed to confirm a block. It also validates that votes from voters are from valid voters in the same way as validation of makers is validated.

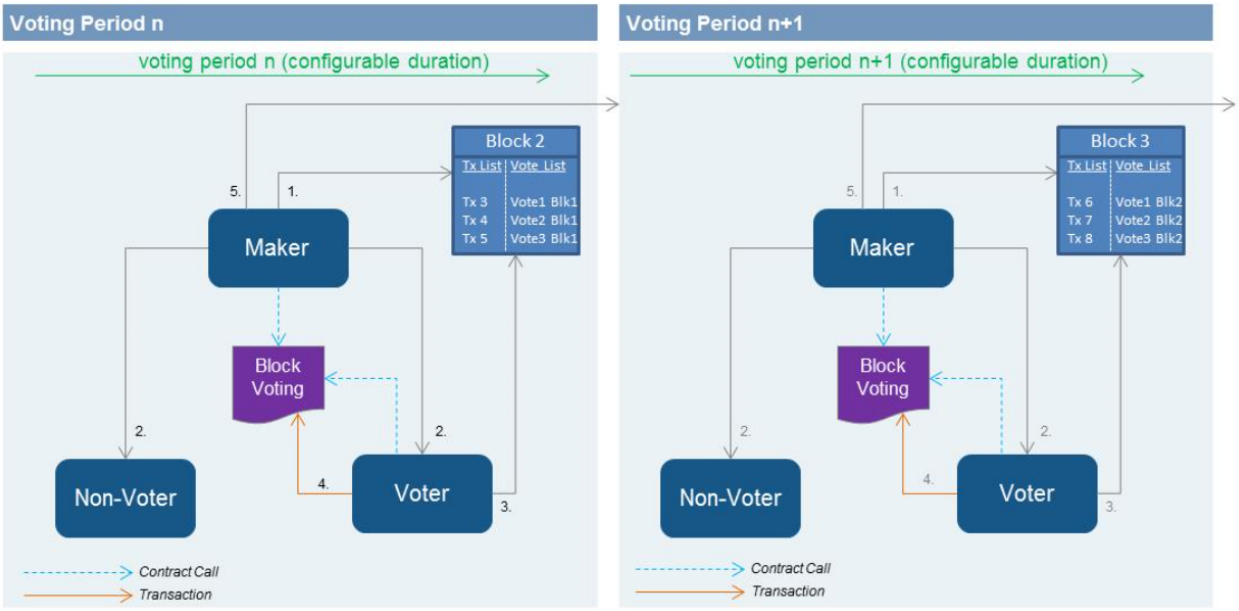
Maker nodes are responsible for minting new blocks and their addresses on the Ethereum network are whitelisted in this smart contract. There must be at a minimum 1 Maker defined for the system to work. The initial Maker nodes are created / defined in the genesis block, but the addition and removal can be done post blockchain creation.

There is a third node type/role in the network, which is the Observer role. These are simply nodes that are not of role Voter or Maker, and will not mint blocks or vote, but simply receive and validate blocks.

Blocks are made by makers, and to avoid multiple Makers creating a block at the same time, each maker will use a random value inside a threshold define for the maker of (min / max) time to create a block. This randomization will eventually mean a maker will recognize it is time for them to create a new block. When a Maker makes a block, other Makers will be notified and reset their timers for when they will

create a block. This is defined in the Voting smart contract defined above and can be set at geth startup via command line parameters, minblocktime and maxblocktime time in seconds.

NOTE: In the current implementation, the Maker is also a Voter to avoid a situation where limited online Voters, could cause chain-halting.



Transaction Processing

Another key differentiator with Quorum is the fact that the chain can include transaction privacy rules. To explain this further, transactions can be thought of as public or private. Its important to understand that Quorum in not introduction a new transaction type to Ethereum but using the existing transaction model for public transactions, and extending Ethereum to support a new (optional) parameter named privateFor, which will be used to designate whom should be able to view this transaction.

First public transactions are transactions in which the payload is visible to all participants of the blockchain or Quorum network. These are exactly the same as transactions used in Ethereum on other networks, no changes.

NOTE: The context here and name “public” transactions does not imply that these transactions are on the public Ethereum network (network 1). These transactions are only on the private Quorum network, the name is used simply to mean that the payloads of these transactions are not private and can be viewed by all participants.

The other type of transaction is private transactions, and these expose their payloads only to the participants on the network whose public keys are specified in the privateFor parameter of the transaction.

NOTE: The privateFor parameter is an array and can contain a list of comma separated keys that should be able to view the transaction.

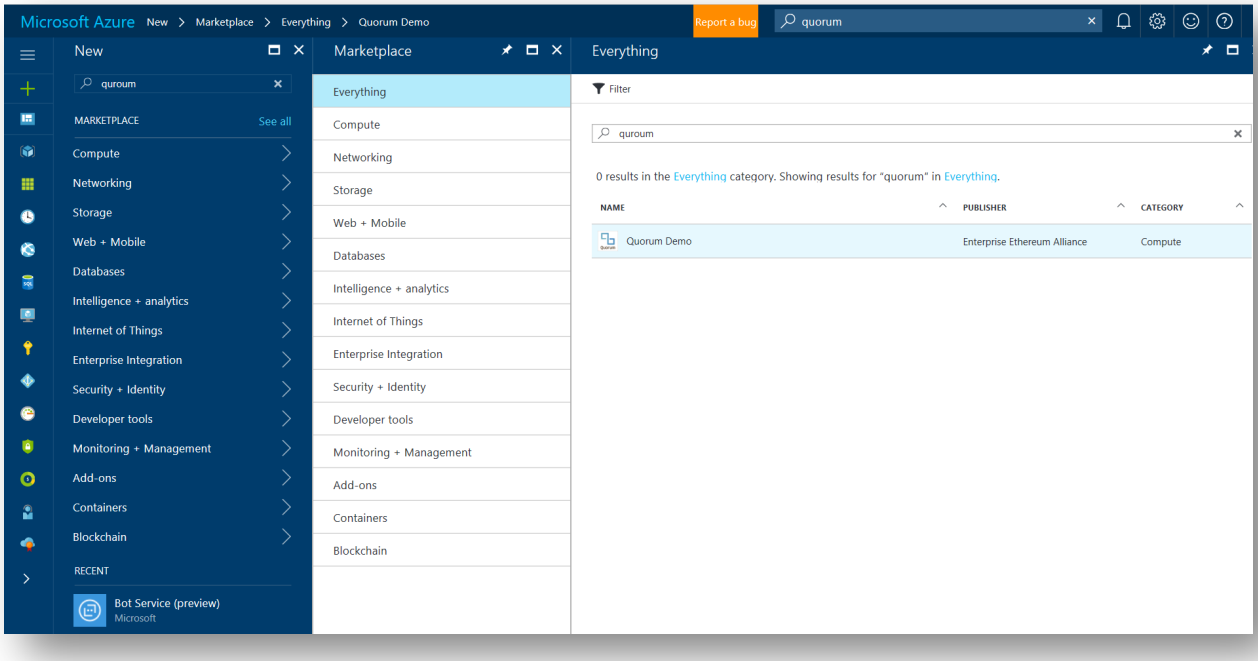
Execution of these types of transactions are different as well. Public transactions are executed in the same way standard Ethereum transactions, each node will execute the contract code and update the state of the contract. For private transactions, things are handled a bit differently. Prior to the Quorum node propagating the transaction to the rest of the network, it replaces the payload with a hash of the encrypted payload that is receives from constellation. Participants on the network that have their public key included in the transaction, will be able to replace the hash with the actual payload (unencrypted) via their instance of constellation, while participants whom are not included in the transaction, will only see the hash. This results in participants who are not party to see the transaction to just skip it, and not perform an operation, those whom are involved will replace the hash with the original payload and have the EVM execute it, and update their state.

This could result in two different states based on whether the participant is party to the transaction or not, and to address this Quorum maintain sync from the public “trie” of state to the private “trie”. The public trie is globally synced, but private trie is not.

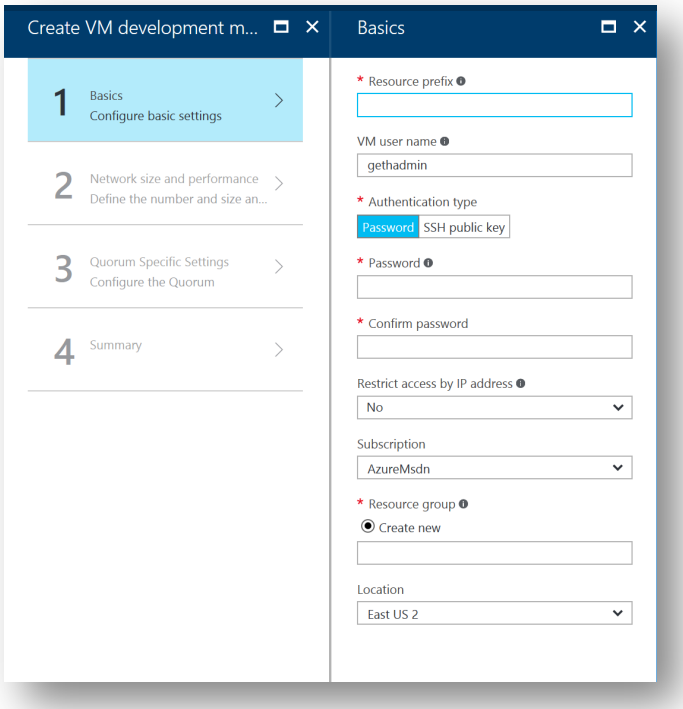
Getting Started

To begin, you will need an Azure subscription that will be used to deploy the virtual machine. If you do not have an Azure subscription, you can create a free Azure account to begin. Because this deployment will only require a single virtual machine, there is no need to increase quotas for your subscription.

Once you have a subscription, go to the Azure portal. Select the '+' symbol in the top left of the portal, and in the pane that appears, in the search box, enter 'Quorum Consortium Network'.



Select the template that is returned in the search results to take you to the Quorum Consortium Network deployment wizard and then click 'Create'. This will open the 'Basics' blade in the wizard.



The template deployment will prompt you for a set of simple inputs to configure the deployment properly. On the first step, the 'Basics' blade, specify the values for standard parameters such as subscription, resource group, and basic virtual machine properties.

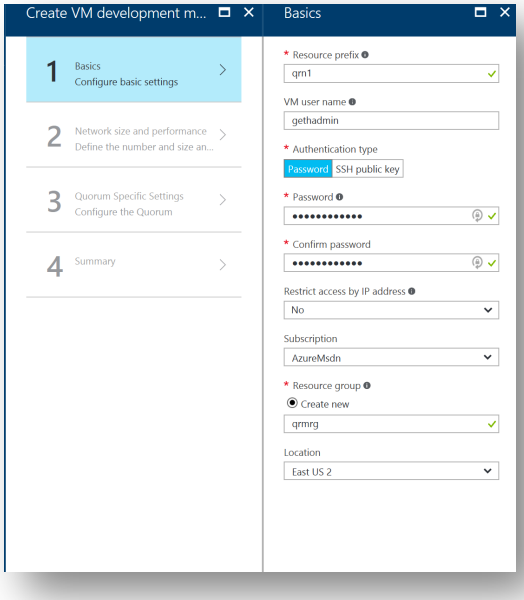
A detailed description of each parameter follows:

Basics

Parameter Name	Description	Allowed Values	Default Value
Resource Prefix	A prefix value that prepended to resource names to ensure uniqueness.	The value is 6 or less alphanumeric characters	NA
VM user name	The username of the administrator for the deployed virtual machine.	The value must be between 1 and 64 characters in length.	NA
Authentication Type	The method used to authenticate to the virtual machine. Username and password or username/ssh key.	Password or SSH public key	SSH public key
Restrict access by IP address	Used to restrict access to all endpoints from a specific IP address or subnet	Yes or No	No
Subscription	The subscription in which to deploy.	Valid subscription	Current subscription
Resource Group	The resource group in which to deploy.	Create new or use existing. For new, the value must be 1 and 90 characters in length.	NA

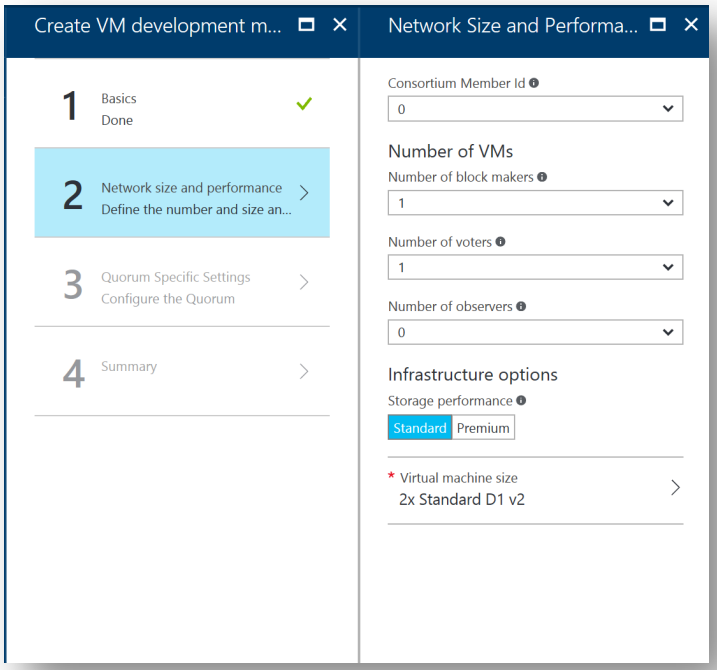
Location	The Azure region in which to deploy.	List of valid Azure regions.	Current region.

A sample deployment is shown below:



Network Size and Performance

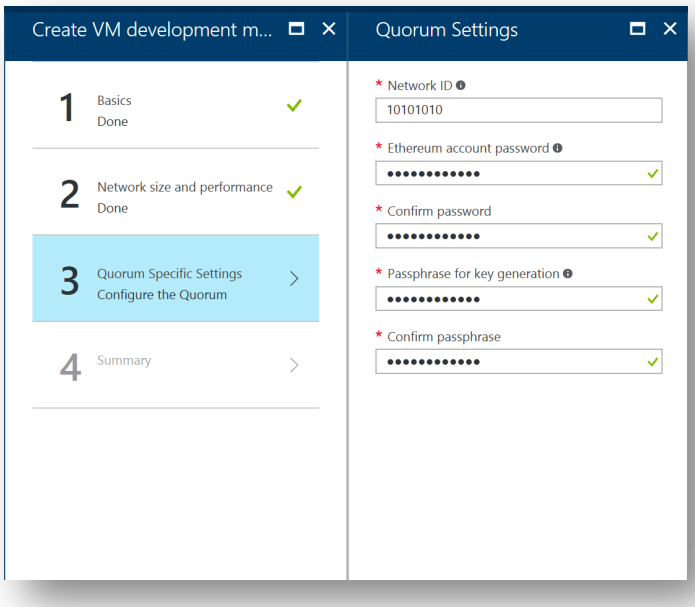
Parameter Name	Description	Allowed Values	Default Value
Consortium Member Id	Set the member id for the participant. In a multi-member network, each member should have a unique id	0-15	0
Number of block makers	Quantity of block maker nodes which will create and propose blocks to the network. (Currently limited to 1)	1	1
Number of voters	Quantity of voter nodes which will validate and vote on block confirmations.	1-9	1
Number of observers	Quantity of observer nodes which will validate transactions only.	0-9	0
Storage performance	Performance of underlying storage (Standard is HDD backed and Premium is SSD)	Standard or Premium	Standard
Virtual Machine Size	Choose the appropriate size of virtual machine.	NA	Recommended sizes are displayed.



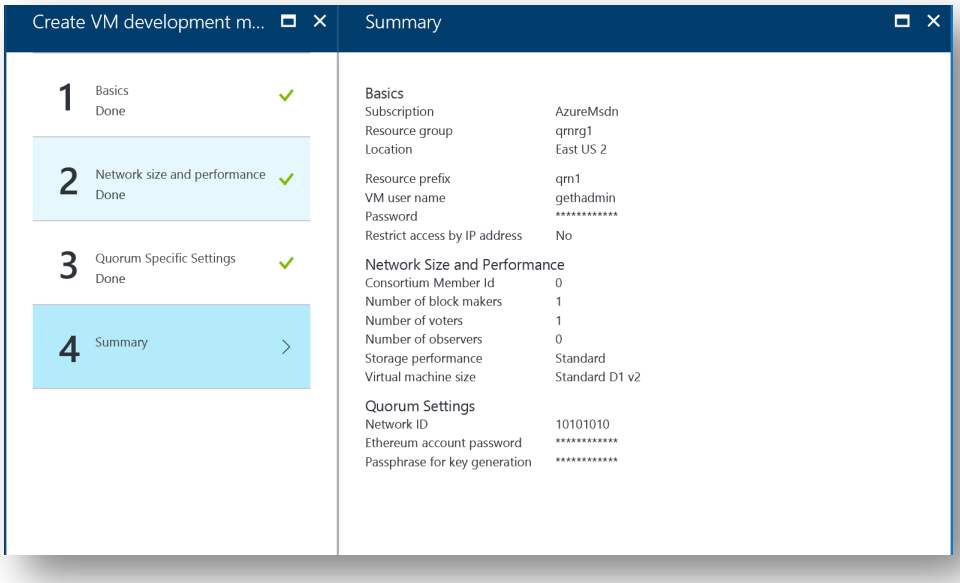
Quorum Specific Settings

Parameter Name	Description	Allowed Values	Default Value
Network ID	ID of the underlying Ethereum network. Peers can only communicate in the same network.	Numeric value up to 9 digits	10101010
Ethereum account password	Password used to secure the default Ethereum account that will be generated	12 characters or more with a minimum of 1 lower case, 1 upper case, and 1 number. Double quotes not allowed.	N/A
Confirm Password	Confirmation of password for Ethereum account	12 characters or more with a minimum of 1 lower case, 1 upper case, and 1 number. Double quotes not allowed.	N/A
Passphrase for key generation	Passphrase used to generate the private key associated with the default Ethereum account that is generated.	12 characters or more with a minimum of 1 lower case, 1 upper case, and 1 number. Double quotes not allowed.	N/A
Confirm Password	Confirmation of passphrase for key generation	12 characters or more with a minimum of 1 lower case, 1 upper case, and 1 number. Double quotes not allowed.	N/A

A sample of this blade is below:



Click through the summary blade, which displays the inputs that have been provided for the deployment of the virtual machine. This also validates the subscription and the inputs to ensure the deployment values will not cause an exception in provisioning.



Finally, review the legal and privacy terms and click ‘Purchase’ to deploy. This typically takes a few minutes to complete.

Post Deployment Quorum Consortium Network

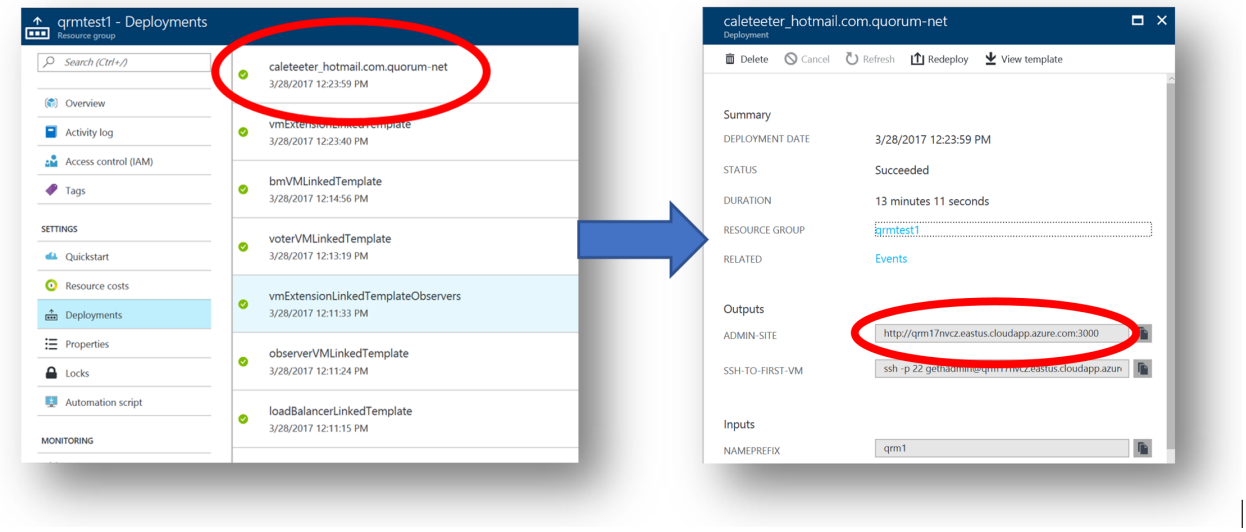
After the deployment of the Quorum Consortium Network, the assets created during the deployment will be fully configured and ready for use. As described above, the infrastructure deployed in Azure will deploy all virtual machines a single virtual network and a single subnet inside that network. A network security group will be provisioned that will provide the following rules:

- Block outbound peer to peer traffic for Ethereum nodes. This isolated the nodes from the public blockchains.
- Allow SSH, administrative web site and cakeshop sites access from public internet.

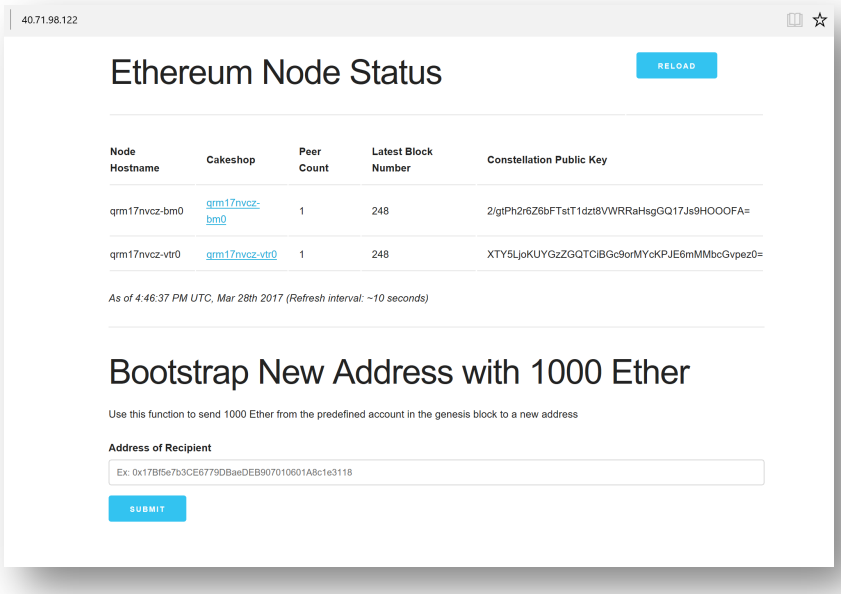
A single load balancer will be deployed which will be the only endpoint exposed to the internet by default and will round robin direct traffic the block makers, voters and observers.

Connect to the Administrative Web Site for the Quorum Network

To view the network nodes that have been created, after deployment the administrative web site will surface the information about the status of these nodes. This can be accessed by navigating to your public ip address for the network. This can be found by reading the output parameters that were generated when the network was created.



Navigate to the site here, which will be named based on the parameters passed at deployment time.

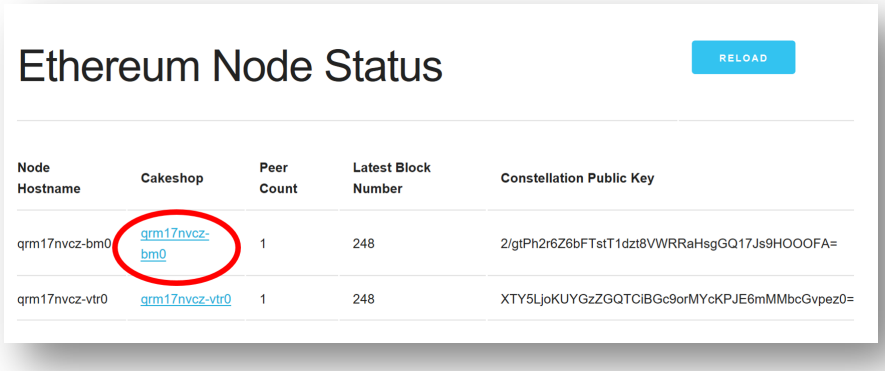


There are several important pieces of information here that should be noted for use later. All nodes are listed on the upper section, along with their hostname, a link to the management interface of cakeshop for the node, the total peers the node is communicating with, the latest block number and the public key for the node (used for private transactions).

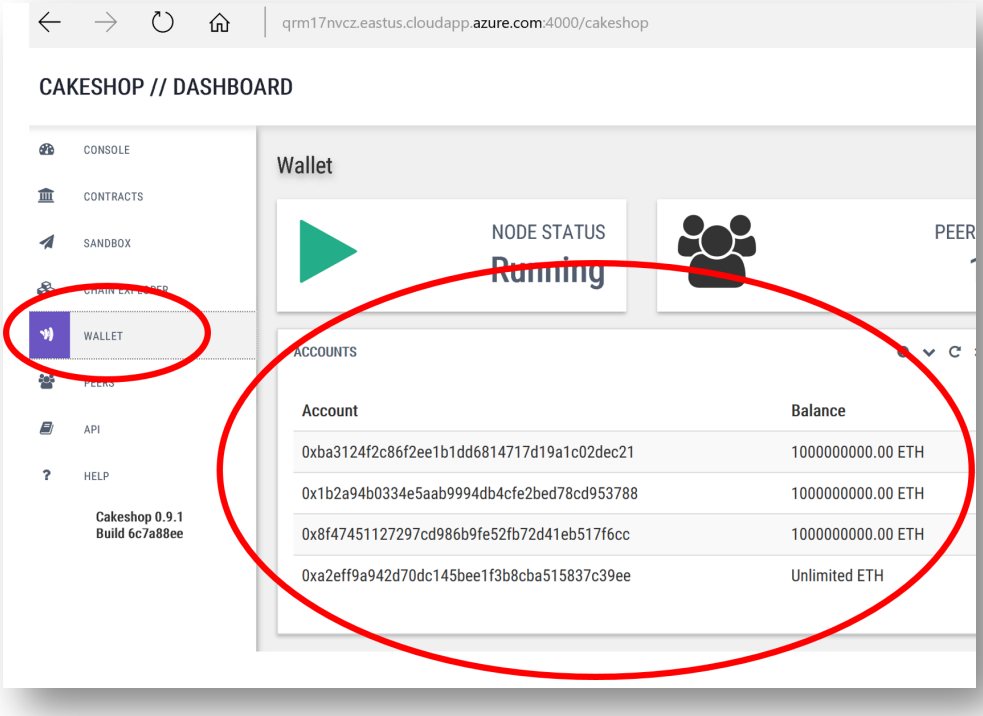
On the bottom half of the interface, you will see a field that will accept a recipient address of an account in the Ethereum network. This is primarily used to "fund" new accounts that are created in the network. This avoids the need to SSH to the nodes and attach to the geth process in order to create/fund new accounts.

Deploying a smart contract and creating private transactions

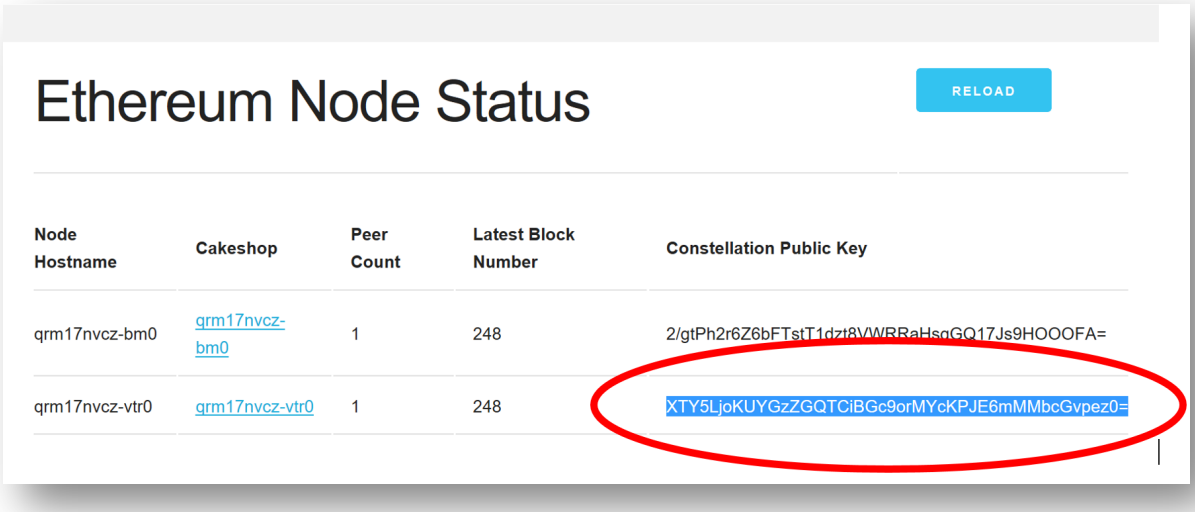
By default, there are several accounts created (Ethereum external accounts). These can be used to interact with the blockchain. To view these accounts, click on the url for cakeshop for the block maker node.



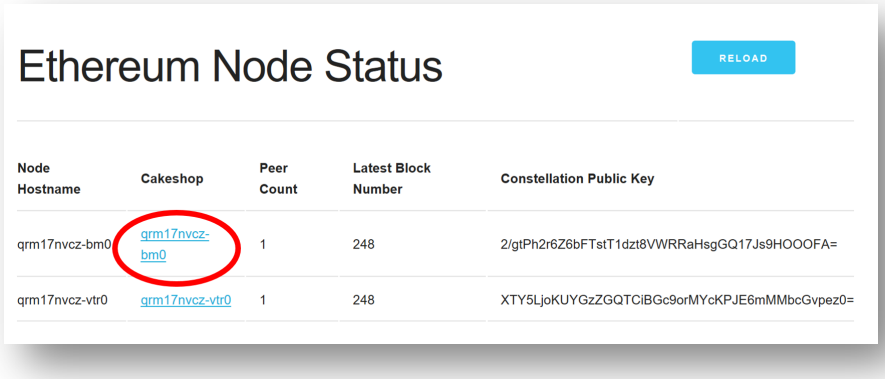
This will open Cakeshop, which is an IDE/monitoring tool created by JP Morgan Chase to view the network based on the node you have clicked on. It displays a variety of information about the network, peers, etc. First, we can view the accounts that exist on our network by clicking on wallets on the left navigation bar.



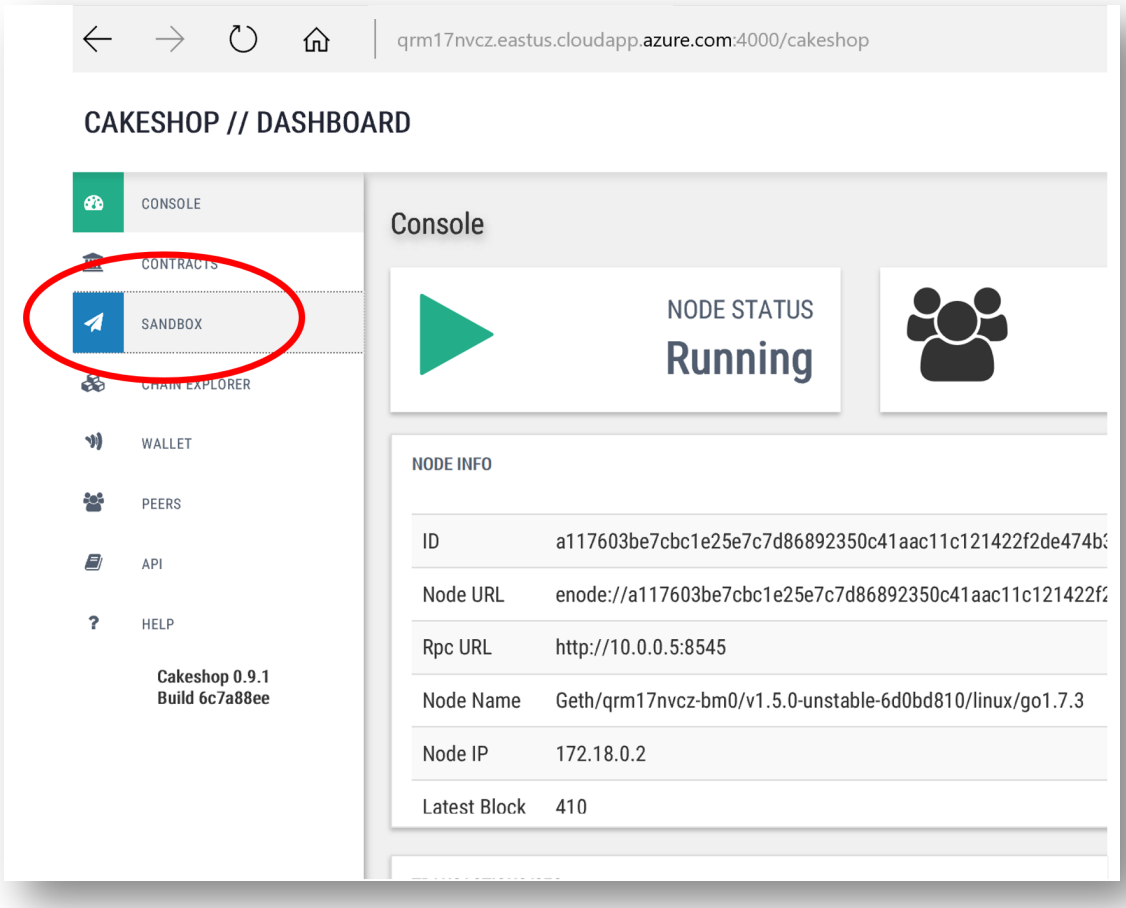
Next we can deploy a smart contract and orchestrate a transaction. First, navigate back to the admin page and copy the public key for a node. In the nodes displayed here, we will copy the public key for the voting node to your clipboard by right click and copy.



Next navigate to the Cakeshop instance on the block maker.

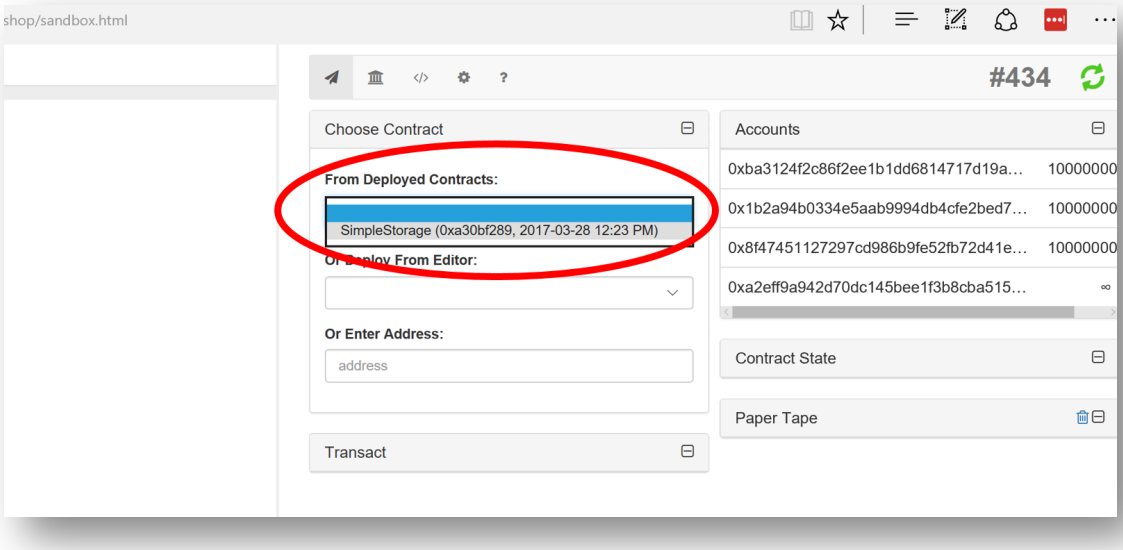


Next navigate to the Sandbox area, which is found on the left navigation pane.

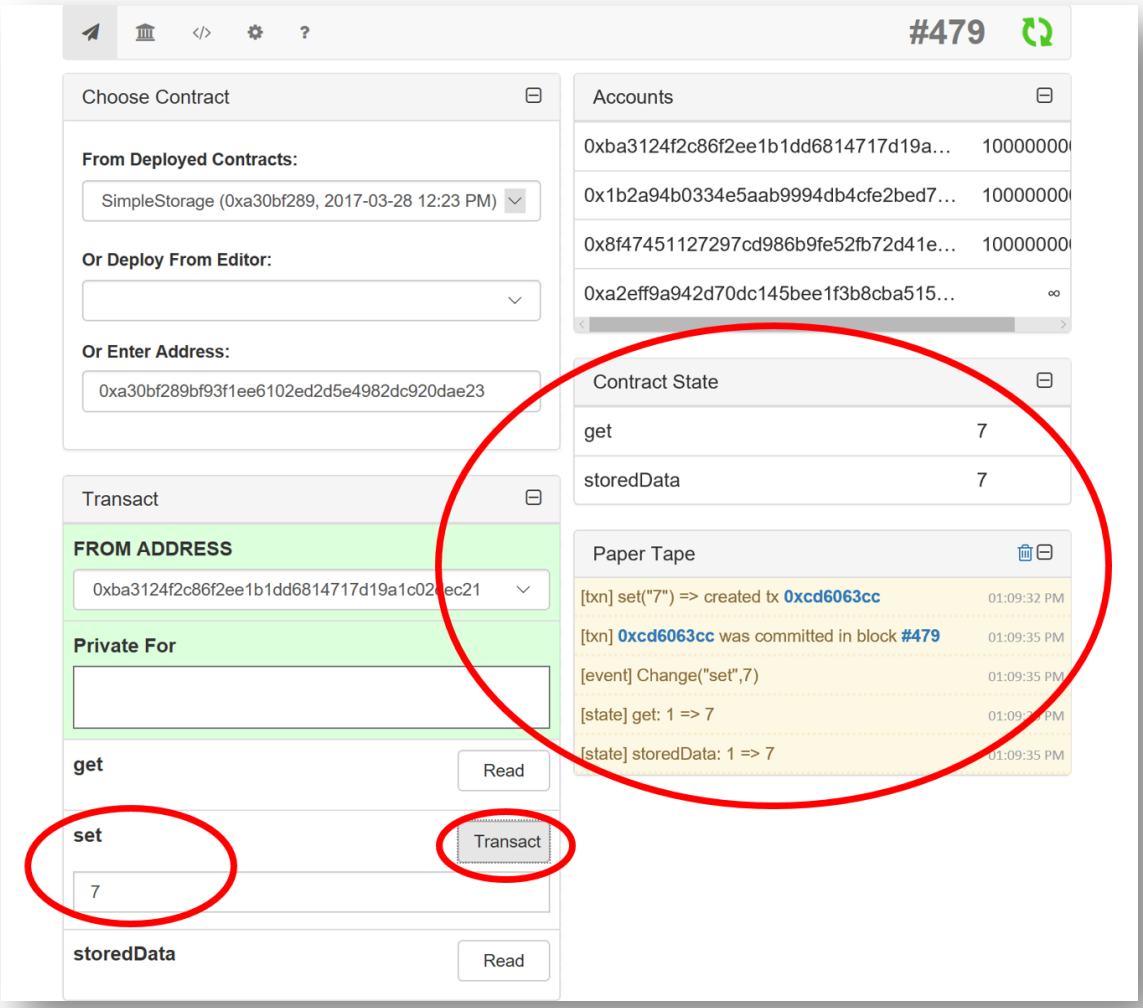


When this tab opens, the interface will display an example smart contract on the left side of the screen and some controls on the right side. We will deploy a new smart contract, using the public key that we

copied in the previous step. On the right side of screen select the dropdown for From Deployed Contracts

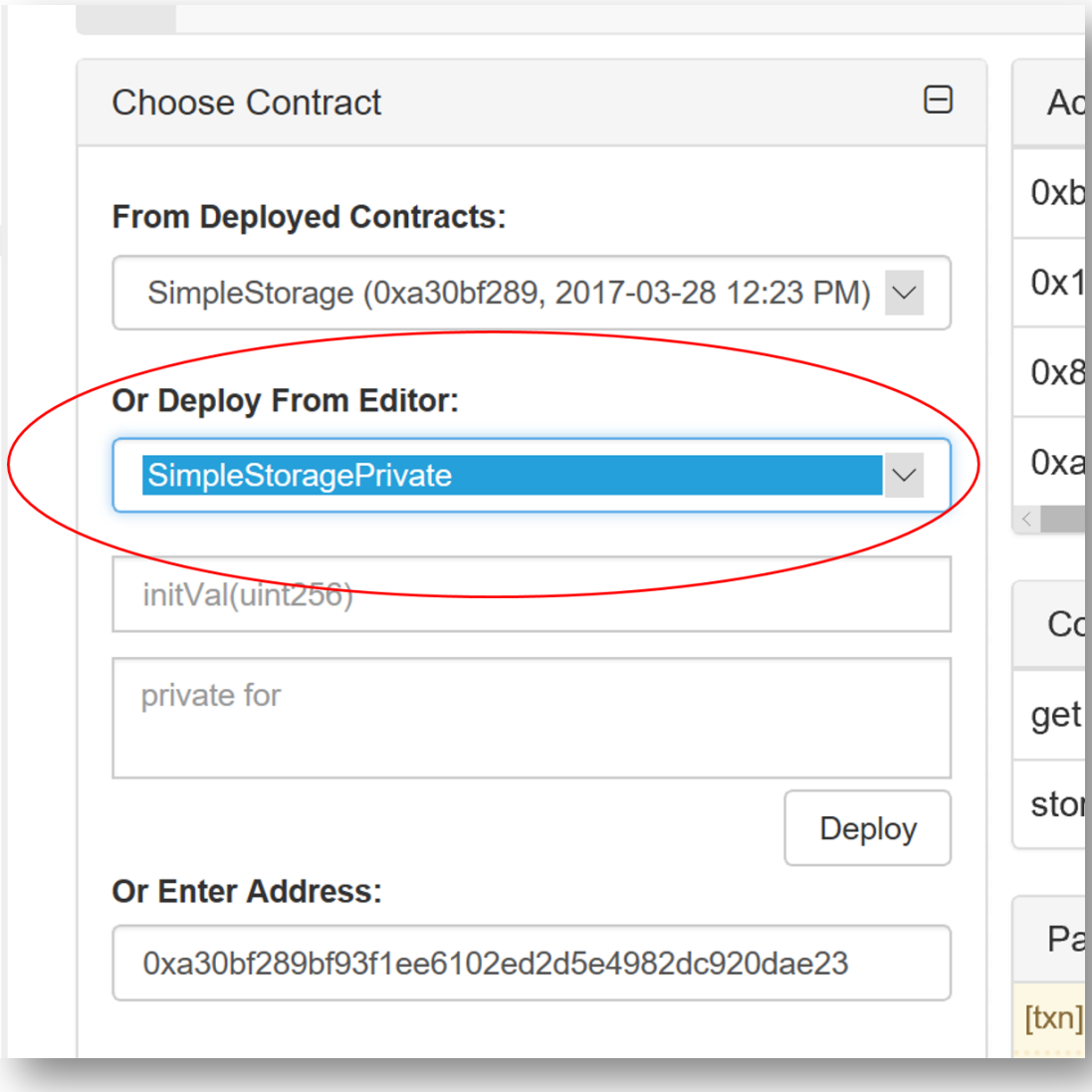


Select the SimpleStorage contract there, and at the bottom fill in an integer for the set function and click transact. This will create a transaction, targeting the SimpleStorage smart contract which was pre-deployed to the blockchain, and the results will be displayed in the Paper Tape section. After a few seconds the transaction should clear and the Contract state will be updated.

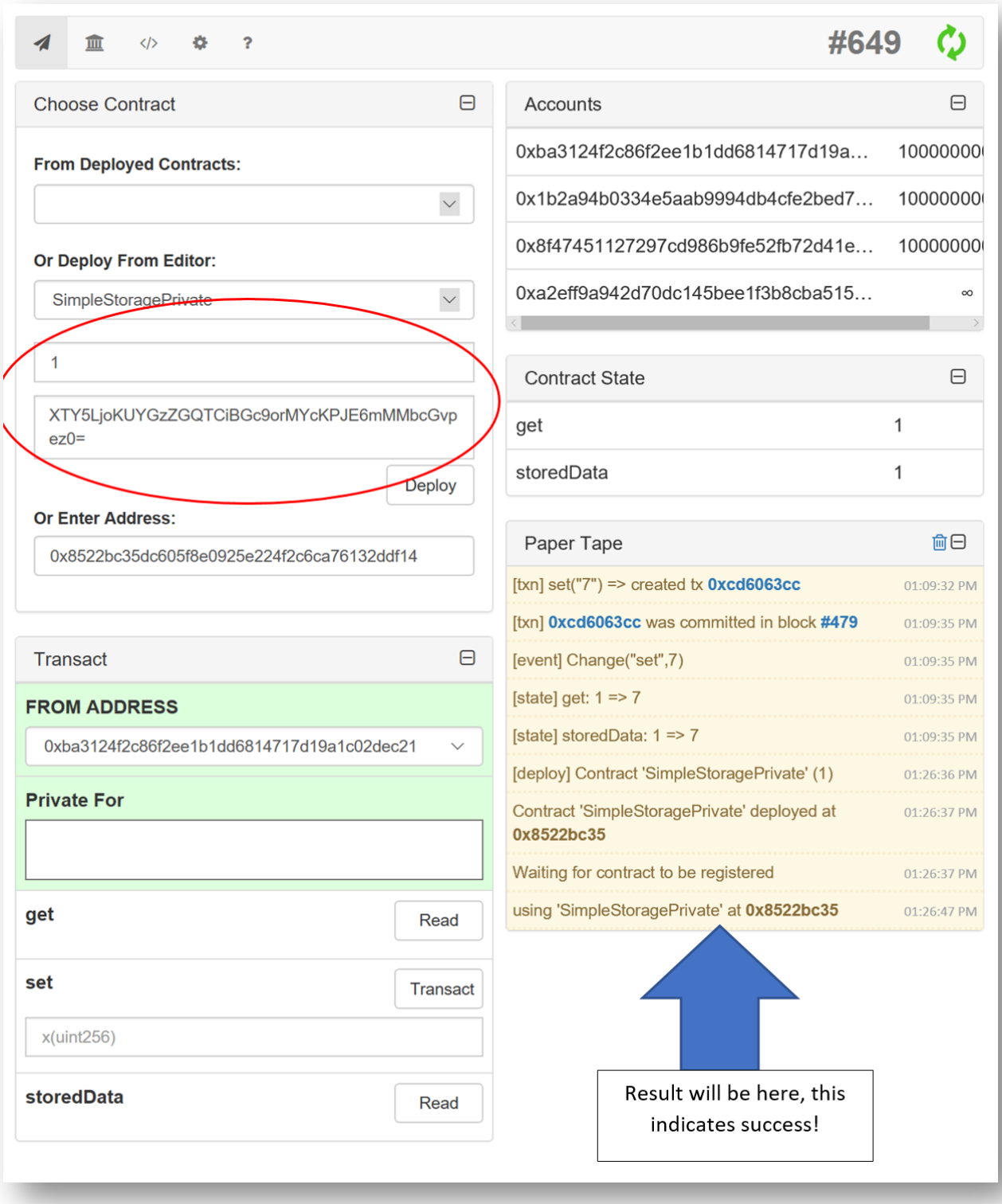


This demonstrates creating a public transaction to an existing smart contract on the Quorum network. Next we can demonstrate deploying a new smart contract and using a private transaction. To start this, on the left side of the screen change the name of the smart contract and constructor.

Next on the right side of the screen, select the dropdown named **Or Deploy From Editor: .** The name of your new smart contract will be shown here.



Directly under this dropdown, 2 textboxes are there to add a value to initialize the state of the smart contract as well as the array of public keys to attach to this. Enter a value in the initVal field (integer) and the public key copied in previous step and click deploy.


#649
Choose Contract
From Deployed Contracts:
Or Deploy From Editor:
SimpleStoragePrivate
1
XTY5LjoKUYGzZGQTCiBGc9orMYcKPJE6mMMbcGvpez0=
Deploy
Or Enter Address:
0x8522bc35dc605f8e0925e224f2c6ca76132ddf14
Transact
FROM ADDRESS
0xba3124f2c86f2ee1b1dd6814717d19a1c02dec21
Private For
get
Read
set
Transact
x(uint256)
storedData
Read
Accounts
0xba3124f2c86f2ee1b1dd6814717d19a…
0x1b2a94b0334e5aab9994db4cfe2bed7…
0x8f47451127297cd986b9fe52fb72d41e…
0xa2eff9a942d70dc145bee1f3b8cba515…
Contract State
get 1
storedData 1
Paper Tape
[txn] set("7") => created tx 0xcd6063cc 01:09:32 PM
[txn] 0xcd6063cc was committed in block #479 01:09:35 PM
[event] Change("set",7) 01:09:35 PM
[state] get: 1 => 7 01:09:35 PM
[state] storedData: 1 => 7 01:09:35 PM
[deploy] Contract 'SimpleStoragePrivate' (1) 01:26:36 PM
Contract 'SimpleStoragePrivate' deployed at 0x8522bc35 01:26:37 PM
Waiting for contract to be registered 01:26:37 PM
using 'SimpleStoragePrivate' at 0x8522bc35 01:26:47 PM
Result will be here, this indicates success!